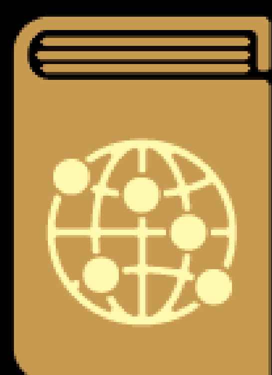


国家级实验教学示范中心联席会计算机学科规划教材
教育部高等学校计算机类专业教学指导委员会推荐教材
面向“工程教育认证”计算机系列课程规划教材

汇编语言教程与实验

刘军 编著



清华大学出版社

国家级实验教学示范中心联席会计算机学科规划教材
教育部高等学校计算机类专业教学指导委员会推荐教材
面向“工程教育认证”计算机系列课程规划教材

汇编语言教程与实验

刘　军　编著

清华大学出版社
北　京

内 容 简 介

汇编语言课程是计算机类专业的一门专业基础课，理论性和实践性非常强。本书将理论教学与实验有机结合，以 8086 CPU 为主，详细介绍汇编语言的基础知识和程序设计方法，主要内容包括：汇编语言基础知识、8086 微型机硬件组织、汇编指令与寻址方式、汇编语言程序格式与数据组织、数据传送程序、算术运算程序、位运算程序、串操作、分支程序设计、循环程序设计、子程序、中断与 DOS 功能调用、宏汇编技术、综合性程序设计案例等。在实验环境上，介绍 DEBUG 和 Masm for Windows 集成实验环境的使用方法。在内容上突出实践教学特色，将实验教学内容贯穿于整个教学过程，每章均附以一定的实验内容。通过多层次的上机实验，加强学生对汇编语言的理解，提高应用编程和程序调试能力。附录部分配有一定量的模拟试题及参考答案，供自我检测使用。

本书可以作为大学本科计算机及相关专业汇编语言课程（含实验环节）的教材或参考书。

图书在版编目（CIP）数据

汇编语言教程与实验/刘军编著. —北京：清华大学出版社，2018
（面向“工程教育认证”计算机系列课程规划教材）
ISBN 978-7-302-47237-7

Ⅰ. ①汇… Ⅱ. ①刘… Ⅲ. ①汇编语言－程序设计－高等学校－教材 Ⅳ. ①TP313

中国版本图书馆 CIP 数据核字（2017）第 122605 号

责任编辑：付弘宇 张爱华
封面设计：刘 键
责任校对：时翠兰
责任印制：刘海龙

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 **邮 编**：100084
社 总 机：010-62770175 **邮 购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：三河市金元印装有限公司
经 销：全国新华书店
开 本：185mm×260mm **印 张**：15.5 **字 数**：375 千字
版 次：2018 年 5 月第 1 版 **印 次**：2018 年 5 月第1 次印刷
印 数：1～1200
定 价：39.00 元

产品编号：075047-01

前　言

汇编语言作为计算机类专业的一门专业基础课，是微机原理与接口技术等课程的重要基础。通过学习汇编语言，能够加深对计算机组成原理的理解，能够感知、体会和理解机器的逻辑功能，向上为理解各种软件系统的原理打下技术理论基础；向下为掌握硬件系统的原理打下实践应用基础。虽然高级语言和软件开发工具在计算机应用领域发展迅速，但汇编语言在底层编程中仍具有其特有的优势。因此，汇编语言仍然是计算机应用领域非常重要的专业技术基础。

虽然汇编语言对计算机硬件依赖性很强，不同的微处理器具有不同的指令系统，但其编程的基本原理相同。因此，本书采用典型的 8086 微处理器汇编语言编写，对于后继 80x86 机型及其他微处理器的汇编语言（例如：51 单片机系列汇编语言、ARM 系列处理器的汇编语言等）具有借鉴和指导意义。

本书是作者在总结 20 多年教学经验和课程改革的基础上编写而成的，在内容编排上，不同于技术手册，而是按照教学规律精心组织教学内容，层层递进，内容全面，重点突出，知识结构层次清晰，例题与实验操作有机结合，书中所有源程序均利用“Masm for Windows 集成实验环境”上机调试通过。本书并没有详尽论述所有的汇编指令和伪指令，而是有重点地选择典型的基本指令和伪指令，并将其分散到各章教学内容之中，使指令与程序设计结合在一起，使得教学内容更加实用，便于学生实际应用，有效地提高学生学习汇编语言的积极性。

根据“加强基础、培养能力、突出实践”的原则，在教学中采用多元化的授课方式。基础知识部分，以课堂教学为主，强调汇编语言的硬件基础知识和程序的基本格式。在汇编指令与寻址方式中，理论与实验相结合，利用 DEBUG 验证理论分析结果，使学生深刻理解指令寻址的内涵。在程序设计方法上，注重精讲设计程序的思路、采用的基本方法和技巧、使用指令的正确方法，将实践环节贯穿于整个教学过程，每章均安排了一定量的实验内容。同时，注意基本知识的融会贯通，力图建立一个完整的知识体系，避免学生割裂前后知识点间的因果关系。

全书内容共分 14 章。第 1 章介绍汇编语言必备的一些基础知识，重点是激发学生学习汇编语言的积极性，让学生在实验中感受汇编语言的特点。第 2 章主要介绍 8086 微处理器基本结构，重点是 CPU 寄存器和存储器结构，掌握逻辑地址、物理地址、偏移地址的概念，并进一步学习 DEBUG 的使用。第 3 章介绍 8086 CPU 指令系统的概况及常用的寻址方式，重点是与数据有关的寻址方式，详细的指令介绍放在了后续章节中。第 4 章介绍汇编语

言源程序格式和数据组织以及相关的伪指令，同时讲述汇编语言的上机操作过程，这是学习汇编语言编程的基础。第5～7章主要介绍数据传送指令、算术运算指令、逻辑运算和移位操作指令及顺序结构程序设计，将指令介绍与程序设计结合在一起，可以提高学生的学习兴趣，避免教学过于枯燥。第8章介绍串操作指令及其程序设计，要求掌握串操作指令的具体格式、功能和用法，能够编写串操作处理程序。第9章介绍分支程序设计，要求重点掌握条件转移指令的功能、分支程序结构，掌握简单分支程序设计和多分支程序设计的方法。第10章介绍循环控制指令和循环程序设计，要求掌握循环程序的基本结构、程序设计及上机调试方法。第11章介绍子程序的定义及其调用，掌握子程序的设计方法及参数传送方式，重点是不同参数传递的子程序设计方法。第12章介绍中断以及常见的DOS功能调用方法。在实际教学中，视需要可以将简单的1号和2号DOS功能调用提前到前面的章节中介绍，让学生尽早体验到汇编语言的输入输出操作。第13章重点介绍宏功能的使用过程。第14章提供了几个典型的综合性程序设计案例，引导学生开展综合性汇编语言程序设计。附录部分除了介绍实验环境、常用命令等，还提供了模拟试题及参考答案，供教师和学生选用。

本课程的教学总学时建议为48～64课时，可以根据实际的教学需要进行适当增减。具体学时分配如下表所示：

建议学时分配表

章　次	章节名称	总课时	授课学时	实验学时
1	基础知识	2	1	1
2	8086微型机硬件组织	4	2	2
3	汇编指令与寻址方式	6	4	2
4	程序格式与数据组织	6	4	2
5	数据传送程序	6	4	2
6	算术运算程序	6	4	2
7	位运算程序	4	2	2
8	串操作	4	2	2
9	分支程序设计	6	4	2
10	循环程序设计	6	4	2
11	子程序	4	2	2
12	中断与DOS功能调用	4	2	2
13	宏汇编技术	2	1	1
14	综合性程序设计案例	4	2	2
合计		64	38	26

为了让老师能较为方便地讲授，本书免费提供所有章节的PPT课件，也提供了书中所有实例的源程序供读者执行和修改。这些配套资料请从清华大学出版社网站www.tup.com.cn下载，下载与使用中的相关问题请联系fuhy@tup.tsinghua.edu.cn。

本书是由天津财经大学刘军教授独立编写完成，体现了作者在多年讲授汇编语言课程的过程中积累的宝贵教学经验，是重点课程建设的重要成果。在编写过程中，汲取了国内外

经典的优秀教材之精华，融入了作者的教学体会并精心组织和编排。本书在编写过程中得到了任春明老师、钟家民老师以及各位同行专家的支持，本书的出版工作也得到了清华大学出版社的大力支持，在此表示衷心的感谢。

尽管作者精心编写，但书中难免有疏漏之处，敬请同行专家和读者指正。作者的电子邮箱：liujun@tjufe.edu.cn。

刘　军

2018年1月



目 录

第1章

基础知识

汇编语言是面向硬件的程序设计语言,只有在具备一定计算机基础知识的基础上去学习,才能有效地应用汇编语言编程。因此,本章主要介绍计算机语言的发展以及汇编语言的特点,介绍数制与信息编码等必备的基础知识,并通过 DEBUG 调试工具初步认识汇编语言。

1.1 计算机语言的发展

1.1.1 计算机语言概述

计算机语言从低级到高级发展,经历了机器语言、汇编语言、高级语言几个阶段,每个阶段的计算机语言都有各自的特点。

1. 机器语言

在计算机发展的初期,由于计算机硬件本身的限制,只能识别二进制代码,于是就使用二进制代码构成机器指令来编写程序,这种二进制编码的计算机语言就是机器语言。机器语言描述的程序称为目标程序,只有目标程序才能被 CPU 直接执行。一条机器指令通常由操作码和操作数两部分组成。其中,操作码指出计算机所进行的具体操作,如加法、减法等;操作数说明操作的对象,如数据或数据存放的地址。机器语言的特点是计算机可直接识别并执行,但依赖于硬件,不同类型机器之间的语言不通用,可移植性差,程序难以理解和调试,指令不便于记忆。

2. 汇编语言

由于机器语言的指令是用二进制代码表示,难以阅读和记忆,给程序设计带来很多困难,于是提出了有助于记忆的符号(或称助记符)来表示二进制代码指令编写程序,这就出现了汇编语言。汇编语言采用助记符表示指令的操作码和操作数,便于阅读、记忆和调试。用汇编语言编写的程序,计算机不能直接识别,必须将其翻译成由机器指令组成的目标程序后,CPU 才能执行,这个翻译过程称为汇编。汇编语言指令与机器语言指令基本上是一一对应的,汇编语言与计算机硬件密切相关,处理器不同,汇编语言就不同。尽管汇编语言还是一种面向机器的语言,不同的 CPU 具有不同的汇编指令,但共性内容是相同的。因此,掌握一种类型的汇编语言,有助于学习其他汇编语言。

3. 高级语言

高级语言类似于自然语言,它采用一组通用的英文单词、数学表达式及规定的符号,按照一定的语法规则和逻辑关系来编写程序。高级语言具有较强的通用性和硬件无关性,是

一种独立于计算机硬件的通用语言。用高级语言编程不必了解和熟悉计算机的指令系统，易学易用。高级语言也要翻译成机器语言才能在计算机上执行，通常有解释型和编译型两种执行方式。

高级语言程序是在未考虑计算机结构特点情况下编写的，经过翻译后的目标程序往往不够精练，过于冗长，加大了目标程序的长度，占用较大存储空间，执行时间较长。

1.1.2 学习汇编语言的必要性

虽然高级语言和软件开发工具在计算机应用领域发展迅速，但汇编语言在底层编程中仍具有其特有的优势。学习汇编语言，有助于深刻理解计算机内部工作机理，更好地对微型计算机系统进行开发和应用。

1. 汇编语言的特点

(1) 汇编语言与计算机硬件密切相关。汇编语言中的指令采用助记符表示，它与机器语言指令基本上是一一对应的，因此它与计算机有着密切的关系，不同类型的 CPU 其汇编语言不同。虽然学习汇编语言比高级语言困难，但它是掌握计算机内部结构的最佳途径。通过汇编语言程序设计，能够更好地理解计算机运行程序的机理，深入了解计算机硬件结构，充分认识计算机。

(2) 汇编语言程序执行效率高。汇编语言是直接在硬件上工作的编程语言，每一条汇编指令都对应着一条机器指令，且汇编语言程序能充分利用计算机硬件特性，如它允许利用寄存器、存储器、标志位等编程。用汇编语言编写的程序在汇编后得到的目标程序效率高，主要体现在空间效率和时间效率上，即目标程序短、运行速度快。在采用相同算法的前提下，任何高级语言程序的效率都远不如汇编语言程序。

2. 汇编语言具有的优势

(1) 能够直接访问与硬件相关的存储器或 I/O 端口。

(2) 能够不受编译器的限制，对生成的二进制代码进行完全的控制。

(3) 能够对关键代码进行更准确的控制，避免因线程共同访问或者硬件设备共享引起的死锁。

(4) 能够根据特定的应用对代码做最佳的优化，提高运行速度。

(5) 能够最大限度地发挥硬件的功能。

(6) 汇编语言用来编制系统软件和过程控制软件，其目标程序占用内存空间少，运行速度快，有着高级语言不可替代的用途。

汇编语言的应用主要是系统程序、效率代码、I/O 驱动程序。某些快速处理、位处理、访问硬件设备等一般都是用汇编语言编写，这正是汇编语言特有的优势。即使在当今计算机语言飞速发展的时代，高级语言也不可能完全替代汇编语言。例如 Linux 内核，虽然绝大部分代码是用 C 语言编写的，但仍然不可避免地在某些关键地方使用了汇编代码。由于这部分代码与硬件的关系非常密切，即使是 C 语言也会显得力不从心，而汇编语言则能够很好地扬长避短，最大限度地发挥硬件的性能。因此，汇编语言仍然是计算机应用领域非常重要的专业技术基础。

1.2　数制与信息编码

1.2.1　数制

按进位方式计数的数制叫作进位记数制，简称进位制。在我们日常生活中，人们习惯于用十进制，即“逢十进一”的数来表示数据。但在计算机内部，数据是以二进制形式表示的，二进制数只有“0”和“1”两个数字，便于用在具有两种稳定状态的物理元件或数字电路中，如双稳态电路来保存二进制信息等。二进制数运算简便，相应的电路设计也十分简单。为了编程者书写数据或输入数据的方便，常用八进制和十六进制。

一个任意的 R 进制数 N，都可写成

$$\begin{aligned} N &= K_n K_{n-1} \cdots K_1 K_0 . K_{-1} K_{-2} \cdots K_{-m} \\ &= K_n \cdot R^n + K_{n-1} \cdot R^{n-1} + \cdots + K_1 \cdot R^1 + K_0 \cdot R^0 \\ &\quad + K_{-1} \cdot R^{-1} + K_{-2} \cdot R^{-2} + \cdots + K_{-m} \cdot R^{-m} \end{aligned} \tag{1-1}$$

式中 m、n 为正整数；R^i 是对应位的位权；R 为 R 进制的基数。所谓基数，就是指在该记数制中每个数位 K_i 可能用到的数字符号的个数，其系数可为 $0 \sim (R-1)$。每个数位计满 R 后就向高位进位，即“逢 R 进一”，在 R 进制数中相邻两个数位的权相差 R 倍，亦即当小数点向左移一位时，数值缩小 R 倍；而当小数点向右移一位时，数值扩大 R 倍。

基数 $R=2$ 时，为二进制数，权为 2^i，K_i 为 0，1 两个数字中的一个，逢二进位。类似地，当基数 $R=8$ 时，为八进制数，权为 8^i；当基数 $R=10$ 时，为十进制数，权为 10^i；当基数 $R=16$ 时，为十六进制数，权为 16^i。

在使用不同进位记数制的数值时，通常在一个数的末尾用一个标识字母来标识。二进制数用字母 B(Binary)，八进制数用字母 O(Octal)或 Q，十进制数用字母 D(Decimal)，十六进制数用字母 H(Hexadecimal)。如果数的尾部没有任何字母，则计算机接收到的数就默认为是十进制数。例如：101101B，127Q，178D 或 178，3CH 等。表 1-1 列出了常用的几种记数制。

表 1-1　常用的记数制

数　　制	基　　数	数　　码
二进制(Binary)	2	0，1
八进制(Octal)	8	0，1，2，3，4，5，6，7
十进制(Decimal)	10	0，1，2，3，4，5，6，7，8，9
十六进制(Hexadecimal)	16	0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F

1.2.2　数制之间的转换

由于计算机采用二进制运算，但用计算机解决实际问题时对数值的输入输出通常使用十进制，这就需要有一个十进制向二进制转换或由二进制向十进制转换的过程。在表达数据或存储单元地址时，经常还会用到八进制和十六进制。这些数制之间的转换关系，是学习汇编语言必备的基础知识。

1. *R* 进制数转换为十进制数

对于任意 R 进制数，按照式(1-1)写成位权展开式，相加求和，就可以得到对应的十进制数。

【例 1-1】 二进制、八进制、十六进制数转换为十进制数。

$$101.11(B)=1\times 2^2+0\times 2^1+1\times 2^0+1\times 2^{-1}+1\times 2^{-2}=5.75$$

$$101(O)=1\times 8^2+0\times 8^1+1\times 8^0=65$$

$$101A(H)=1\times 16^3+0\times 16^2+1\times 16^1+10\times 16^0=4122$$

2. 十进制数转换为 *R* 进制数

将十进制数转换为 R 进制数时，将整数部分和小数部分分别进行转换，然后合并起来即可。

整数部分：采取除以 R 取余法，即将十进制整数不断地除以 R 取余数，直到商为 0，余数从右到左排列，首次得到的余数在最右面，最后得到的余数在最左面。

小数部分：采取乘以 R 取整法，即将十进制小数不断地乘以 R 取整数，直到小数部分为 0 或达到一定精度时为止，整数从左到右排列，首次得到的整数在最左面，最后得到的整数在最右面。

【例 1-2】 十进制数 49.345 转换为二进制数。

(1) 整数部分：除以 2 取余，得到二进制数 110001。

除数	被除数/商	余数
2	49	
2	24	1
2	12	0
2	6	0
2	3	0
2	1	1
	0	1

(2) 小数部分：乘以 2 取整，得到二进制小数 0.01011。

运算	整数
0.345 × 2	
0.690	0
× 2	
1.380	1
× 2	
0.760	0
× 2	
1.520	1
× 2	
1.040	1

(3) 合并结果，即 49.345(D)= 110001.01011(B)。

3. 二进制、八进制、十六进制数之间的转换

由于八进制数、十六进制数与二进制数之间有固定的对应关系：$2^3=8$，$2^4=16$，即一位八进制数可以用 3 位二进制数表示；一位十六制数可以用 4 位二进制数表示，所以可以采用数位对应法进行转换。表 1-2 列出了几种数制之间的对应关系。

表 1-2 不同数制对应关系

十进制	二进制	八进制	十进制	二进制	十六进制	十进制	二进制	十六进制
0	000	0	0	0000	0	8	1000	8
1	001	1	1	0001	1	9	1001	9
2	010	2	2	0010	2	10	1010	A
3	011	3	3	0011	3	11	1011	B
4	100	4	4	0100	4	12	1100	C
5	101	5	5	0101	5	13	1101	D
6	110	6	6	0110	6	14	1110	E
7	111	7	7	0111	7	15	1111	F

1）二进制与八进制数的相互转换

按照数制之间的对应关系，二进制转换为八进制时，以小数点为界向左右两边分组，每3位为一组，不足3位时补0；八进制转换为二进制时，每一位八进制数用对应3位二进制数代替即可。

【例 1-3】 二进制数 1101101110.10011 转换为八进制数。

$\underline{001}\ \underline{101}\ \underline{101}\ \underline{110}.\underline{100}\ \underline{110}=1556.46(O)$

1 5 5 6 4 6

【例 1-4】 八进制数 5124 转换为二进制数。

$5124(Q)=\underline{101}\ \underline{001}\ \underline{010}\ \underline{100}\ (B)$

2）二进制与十六进制数的相互转换

按照数制之间的对应关系，二进制转换为十六进制时，以小数点为界向左右两边分组，每4位为一组，不足4位时补0；十六进制转换为二进制时，每一位十六进制数用对应4位二进制数代替即可。

【例 1-5】 二进制数 1101101110.110101 转换为十六进制数。

$\underline{0011}\ \underline{0110}\ \underline{1110}.\underline{1101}\ \underline{0100}=36E.D4(H)$

3 6 E D 4

【例 1-6】 十六进制数 2A1D 转换为二进制数。

$2A1D(H)=\underline{0010}\ \underline{1010}\ \underline{0001}\ \underline{1101}\ (B)$

1.2.3 二进制数的运算

1. 算术运算

二进制数的算术运算包括加法、减法、乘法和除法四则运算，其运算规则简单易学，分别列举如下。

1）加法规则

0+0=0　　0+1=1

1+0=1　　1+1=0(向高位进位 1)

2）减法规则

0－0=0　　0－1=1(向高位借位 1)

1－0=1　　1－1=0

3）乘法规则

0×0＝0　　　0×1＝0

1×0＝0　　　1×1＝1

4）除法规则

0÷0（无意义）　　　0÷1＝0

1÷0（无意义）　　　1÷1＝1

2. 逻辑运算

二进制数的逻辑运算是按位进行且相互独立的，基本的逻辑运算有与、或、非三种，还有一种经常使用的是异或运算。下面分别加以简单介绍。

1）逻辑与运算

逻辑与也称逻辑乘，表示当A、B事件同时都为真时，结果才为真。其真值表如表1-3所示。

表1-3　与运算真值表

A	B	F＝A×B
0	0	0
0	1	0
1	0	0
1	1	1

2）逻辑或运算

逻辑或运算，表示当A、B事件只要有一个为真时，结果就为真。其真值表如表1-4所示。

表1-4　或运算真值表

A	B	F＝A＋B
0	0	0
0	1	1
1	0	1
1	1	1

3）逻辑非运算

逻辑非运算，表示同原事件A含义相反。其真值表如表1-5所示。

表1-5　非运算真值表

A	$F=\overline{A}$
0	1
1	0

4）逻辑异或运算

逻辑异或运算，表示当A、B事件取值相同时，结果就为假；只有当A、B事件取值不同时，结果才为真。其真值表如表1-6所示。

表 1-6　异或运算真值表

A	B	F=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

1.2.4　机器数的表示方法

在计算机中，通常把一个数的最高位定义为符号位，用“0”表示正，“1”表示负，其余各位表示数值，这种数称为机器数。机器数通常用原码、反码和补码三种方式表示。任何正数的原码、反码和补码的形式完全相同，负数则各自有不同的表示形式。

1. 原码

对于整数的原码，其符号位 0 表示正，1 表示负，数值位是数的绝对值。例如：

$[+3]_{原}$ =0 0000011B= 03H

$[-3]_{原}$ =1 0000011B= 83H

$[+127]_{原}$ =0 1111111B= 7FH

$[-127]_{原}$ =1 1111111B= FFH

2. 反码

对于正数，其反码与原码相同；对于负数，其符号位为 1，数值位是数的绝对值取反。例如：

$[+3]_{反}$ =0 0000011B= 03H

$[-3]_{反}$ =1 1111100B= FCH

$[+127]_{反}$ =0 1111111B= 7FH

$[-127]_{反}$ =1 0000000B= 80H

3. 补码

对于正数，其补码与原码、反码均相同；对于负数，其符号位为 1，数值位是数的绝对值取反再加 1 形成。例如：

$[+3]_{补}$ =0 0000011B= 03H

$[-3]_{补}$ =1 1111101B= FDH

$[+127]_{补}$ =0 1111111B= 7FH

$[-127]_{补}$ =1 0000001B= 81H

对于 0 的表示方法，原码和反码的 0 都不唯一，只有补码的 0 才有唯一值。即 $[+0]_{补}$ = $[-0]_{补}$ =0 0000000B=00H。所以，在微型机中，负数一般用补码表示。汇编语言中涉及的带符号数，也都是以补码形式表示。利用补码，可以方便地进行二进制的算术运算。

在二进制运算中，由于运算结果超出了规定的位数，最高有效位向前的进位，这一位自然丢失，一般不表示结果的对错。而当运算结果超出了字长允许表示的范围，一般会造成结果出错，这叫作溢出。因此，用 8 位补码表示的数为－128～＋127，用 16 位补码表示的数为－32 768～＋32 767。

1.2.5 十进制数的编码

十进制数的二进制编码称为 BCD 码,最常用的一种是 8421BCD 码,每一位十进制数用 4 位二进制数表示,如表 1-7 所示。

表 1-7 8421BCD 码表

十 进 制	BCD	十 进 制	BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

BCD 码在存储时,有压缩 BCD 码和非压缩 BCD 码两种形式。压缩 BCD 码,每个字节存放两位 BCD 码;非压缩 BCD 码,每个字节存放一位 BCD 码,存放在字节的低四位。

1.2.6 字符编码

计算机中通常采用的字符编码是美国信息交换标准代码(American Standard Code Information Interchange,ASCII),它是最主要的字符编码方式。对字符进行编码是进行非数值处理、实现输入输出的基本手段。

标准的 ASCII 码,在一个字节中用 7 位二进制表示字符编码,用一位(最高位)表示奇偶校验位(Parity bit)。在 ASCII 码表(见附录 C)中可以看出,数字 0～9 的编码是 30H～39H,大写字母 A～Z 的编码是 41H～5AH,小写字母 a～z 的编码是 61H～7AH。了解 ASCII 码表的字符分布规律,有助于字符转换程序的编程。

1.3 DEBUG 初步

DEBUG 是汇编语言最基础的调试工具,它不仅可以修正汇编语言程序中的错误,而且可以用来编写较短的汇编语言程序。对初学者而言,DEBUG 更是比较好的入门工具。DEBUG 操作简单,可以避免一开始就碰到许多难以理解的程序行。DEBUG 还可用来检查和修改内存位置、载入存储和执行程序、检查和修改寄存器等。DEBUG 通过单步、设置断点等方式为汇编语言程序员提供了有效的调试手段。

1.3.1 DEBUG 基础知识

所有 DEBUG 命令由一个英文字母,后跟一个或多个参数组成。使用时应该注意:

(1) 用于命令的字母不区分大小写;

(2) 只使用十六进制数,但没有后缀字母 H;

(3) 分隔符(空格或逗号)只在两个数值之间是必需的,命令和参数间可以没有分隔符;

(4) 每个命令只有按了回车键后才有效,可以用 Ctrl + Break 键中止命令的执行;

(5) 如果命令不符合 DEBUG 的规则,则将以 error 提示,并用“^”指示错误所在的

位置。

许多命令的参数是主存逻辑地址，形式是："段基地址：偏移地址"。其中，段基地址可以是段寄存器或数值；偏移地址是数值。如果不输入段地址，则采用默认值，即默认段寄存器值；如果没有提供偏移地址，则通常就是当前偏移地址。

对主存操作的命令还支持地址范围参数，它的形式是："开始地址 结束地址"（结束地址不能具有段地址），或者是"开始地址 L 字节长度"。

启动 DEBUG 的方法有几种：

方法 1：在 Windows XP 的"开始"菜单中选择"运行"命令，在"运行"对话框中输入 DEBUG 并回车（即按 Enter 键），如图 1-1 所示。



图 1-1　"运行"对话框

方法 2：在"运行"对话框中输入 CMD，启动 DOS 命令窗口，在 DOS 命令提示符下输入 DEBUG 并回车。

方法 3：在 Windows XP 的"开始"菜单中，选择"所有程序"|"附件"|"命令提示符"命令，启动 DOS 命令窗口，在 DOS 命令提示符下输入 DEBUG 并回车。

启动 DEBUG 后，出现提示符"-"，窗口形式如图 1-2 所示，此时就可以开始使用 DEBUG 命令，详细的 DEBUG 命令见附录 A。



图 1-2　DEBUG 窗口

1.3.2　用 DEBUG 运行程序

为了使学生快速入门并对汇编语言有一个感性认识，下面在 DEBUG 下运行一个简单的汇编语言程序。

【例 1-7】　在 DEBUG 下，通过运行下面的汇编语言程序。

```
MOV   DL,35H
MOV   AH,2
INT   21H
INT   20H
```

操作步骤：

(1) 启动 DEBUG 后，出现提示符"—"。

(2) 用 A 命令汇编。

```
-A
136E:0100 MOV DL,35
136E:0102 MOV AH,2
136E:0104 INT 21
136E:0106 INT 20
136E:0108
-
```

(3) 用 G 命令运行,可以看到输出一个字符'5',这就是这段程序的功能。

```
-G
5
Program terminated normally
-
```

(4) 用 U 命令反汇编,可以看到汇编指令与机器指令的对应关系。

```
-U
136E:0100 B235          MOV     DL,35
136E:0102 B402          MOV     AH,02
136E:0104 CD21          INT     21
136E:0106 CD20          INT     20
```

(5) 用 Q 命令退出 DEBUG。

在本例题中,35H 就是字符'5'的 ASCII 码,如果换成其他可显示字符的 ASCII 码,则程序就会输出其对应的字符。

1.4 实验内容

【实验目的】 通过在 DEBUG 下运行汇编语言程序,初步了解 DEBUG 的使用,了解汇编语言的特点。

【实验 1-1】 在 DEBUG 下,编写输出字符'A'的汇编语言程序,查看运行结果,并用 U 命令查看机器代码与助记符的关系,说明该程序的功能并总结本次实验的体会。

参考汇编语言程序:

```
MOV   DL,41H
MOV   AH,2
INT   21H
INT   20H
```

习 题

1. 将下列十进制数分别转换为二进制、八进制和十六进制数。

(1) 100D　　(2) 39.125D　　(3) 255D

2. 将下列数分别转换为对应的十进制数。

(1) 2A5H　　(2) 177Q　　(3) 1101110.11B

3．将下列十六进制数分别转换为二进制和八进制数。

(1) 3DH　　(2) 2A7.C6H　　(3) FFH

4．将下列二进制数分别转换为八进制和十六进制数。

(1) 10101100B　　(2) 10111.101B　　(3) 110110010.100101B

5．将下列十进制数分别用 8 位二进制数的原码、反码和补码表示。

(1) +0　　(2) −0　　(3) +53　　(4) −53　　(5) −120

6．已知字母'A'的 ASCII 码是 65，则字母'D'的 ASCII 码是什么？小写字母'a'与大写字母'A'的 ASCII 码相差多少？

7．对于二进制数 10010101 和 01000010，在下列情况下分别相当于十进制数的多少？

(1) 将其看作无符号数　　(2) 将其看作原码

(3) 将其看作反码　　(4) 将其看作补码

第 2 章

8086 微型机硬件组织

汇编语言与微型计算机的硬件密切相关,掌握必要的硬件基础知识是进一步学习的关键。本章主要介绍 8086 微处理器基本结构,重点是 CPU 寄存器和存储器结构,了解各寄存器的工作特点,掌握逻辑地址、物理地址、偏移地址的概念,并通过 DEBUG 查看寄存器和存储器的内容。

2.1 微型计算机概述

2.1.1 微型计算机的基本结构

计算机系统的硬件是由运算器、控制器、存储器、输入设备、输出设备五大部件组成,其中运算器和控制器合在一起称为中央处理器(Central Processing Unit,CPU)。在微型计算机中,运算器和控制器集成在了一块芯片上,称为微处理器,它是微型计算机的核心部件,它的性能决定了整个微型计算机的各项关键指标。存储器包括随机存储器(Random Access Memory,RAM)和只读存储器(Read Only Memory,ROM)。I/O 接口用来连接外部设备和微型计算机。总线为 CPU 和其他部件之间提供数据、地址和控制信息的传输通道。微型计算机的基本结构如图 2-1 所示。

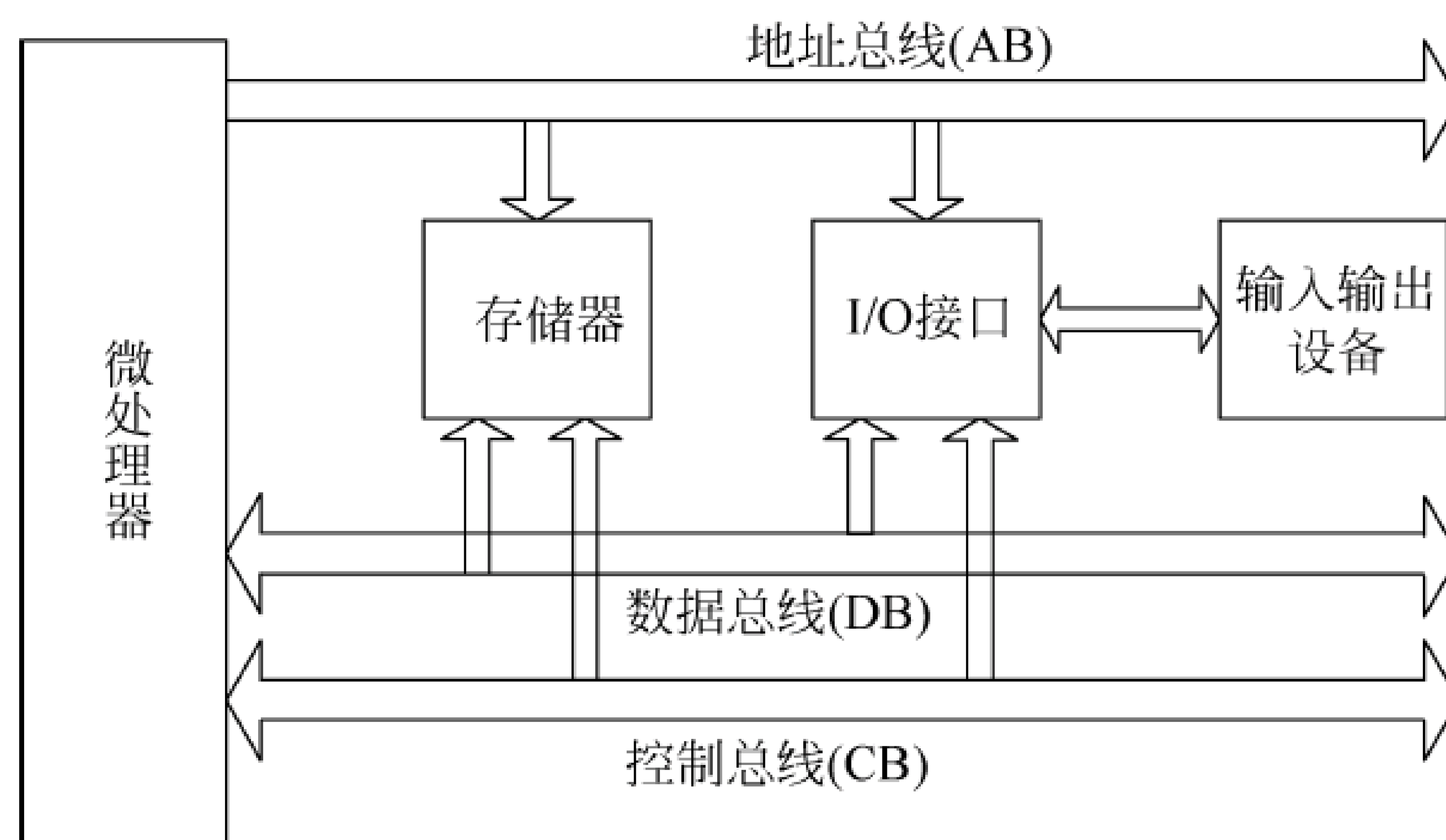


图 2-1 微型计算机的基本结构

微型计算机的系统总线分别是数据总线、地址总线和控制总线。数据总线(Data Bus,DB)用来在微处理器与存储器或其他部件之间进行数据传送,是双向总线,数据总线的位宽决定了 CPU 和外界的数据传送速度,8086 微处理器是 16 位数据总线。地址总线(Address Bus,AB)专门用来传送地址信息,它是单向的,地址总线的位数决定了 CPU 可以直接寻址的内存范围,8086 微处理器是 20 位地址总线,最大寻址范围是 1MB。控制总线(Control

Bus,CB)用来传输控制信号,其中包括微处理器送往存储器和输入输出接口电路的控制信号,如读信号、写信号和中断响应信号等；也包括其他部件送到 CPU 的信号,如时钟信号、中断请求信号和准备就绪信号等。

2.1.2　微处理器

自 20 世纪 70 年代开始出现微型计算机以来,微处理器经历了飞速的发展。1971 年,Intel 公司设计成功了第一片 4 位微处理器 Intel 4004；随之又设计生产了 8 位微处理器 8008；1973 年推出了 8080；1974 年基于 8080 的个人计算机(Personal Computer,PC)问世,Microsoft 公司的创始人 Bill Gates 为 PC 开发了 BASIC 语言解释程序；1977 年 Intel 推出了 8085。自此之后,Intel 又陆续推出了 8088、8086、80286、80386、80486、Pentium 等 80x86 系列微处理器。

8088 微处理器,具有多个 16 位的寄存器、8 位数据总线和 20 位地址总线,可以寻址 1MB 的内存。虽然这些寄存器一次可以处理 2B,但数据总线一次只能传送 1B,属于准 16 位的微处理器。该处理器只能工作在实模式。

8086 微处理器,指令系统与 8088 完全相同,具有多个 16 位的寄存器、16 位数据总线和 20 位地址总线,可以寻址 1MB 的内存,一次可以传送 2B,是 16 位的微处理器。8086 与 8088 两者指令系统相同,汇编语言一致,除了运行速度外,其他方面无区别。该处理器也是只能工作在实模式。

8086 微处理器由执行单元(Execution Unit,EU)和总线接口单元(Bus Interface Unit,BIU)组成,其内部结构如图 2-2 所示。

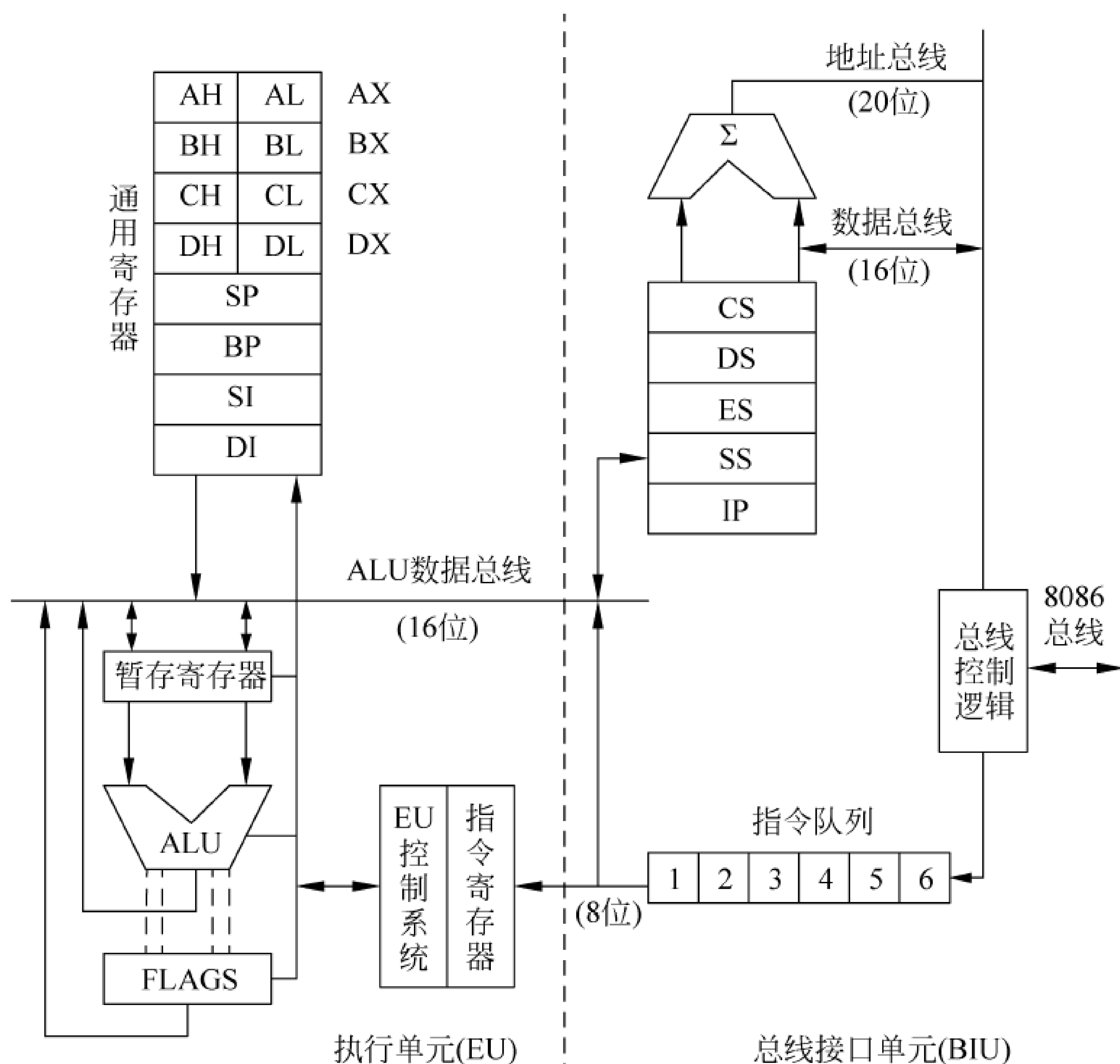


图 2-2　8086 微处理器内部结构

EU 部件主要由算术逻辑单元(Arithmetic Logic Unit,ALU)、标志寄存器、通用寄存器、暂存寄存器和控制电路等组成,负责指令的执行。它从 BIU 的指令队列中取出指令、分析指令并执行指令,而执行指令过程中所需要的数据和执行的结果,也都由 EU 向 BIU 发出请求,再由 BIU 对存储器或外设进行存取操作来完成。

总线接口单元(Bus Interface Unit,BIU)包含一个地址加法器(形成 20 位的物理地址)、一组 16 位的段寄存器、一个 16 位的指令指针 IP、一个 6B 的指令队列缓冲器及总线控制电路。BIU 负责 8086 CPU 与存储器和外设之间的信息传送。具体地说,BIU 负责从内存的指定区域取出指令,送至指令队列排队。在执行指令时所需要的操作数,也由 BIU 从内存的指定区域取出,传送给执行部件 EU 去执行。

EU 和 BIU 能独立运行,在一条指令的执行过程中,就可取下一条指令送入指令队列,实现流水操作,提高了指令的运行速度。

随着微处理器的发展,出现了 80286 微处理器,它比 8086 运行更快,具有多个 16 位的寄存器、16 位数据总线和 24 位地址总线,可以寻址 16MB 内存。它既可以工作在实模式,也可以工作在保护模式。随后出现的 80386/80486 微处理器、Pentium 系列微处理器,寄存器都是 32 位的,在性能和功能上都大大增强。

实际上 80x86 系列的功能还在不断改进和增强,它们的速度将会更快,性能将会更优越。但无论怎样变化,它们总会被设计成是完全向下兼容的,就像在 8086 上设计和运行的软件可以不加任何改变地在 Pentium 4 机上运行一样。对于汇编语言编程人员来讲,掌握 16 位计算机的编程十分重要,它是学习高档计算机及保护模式编程的基础,也是掌握实模式程序设计的唯一方法。

2.2 8086 寄存器组

8086 微处理器中包含四个数据寄存器、四个地址寄存器、四个段寄存器和两个控制寄存器。寄存器组结构如图 2-3 所示。

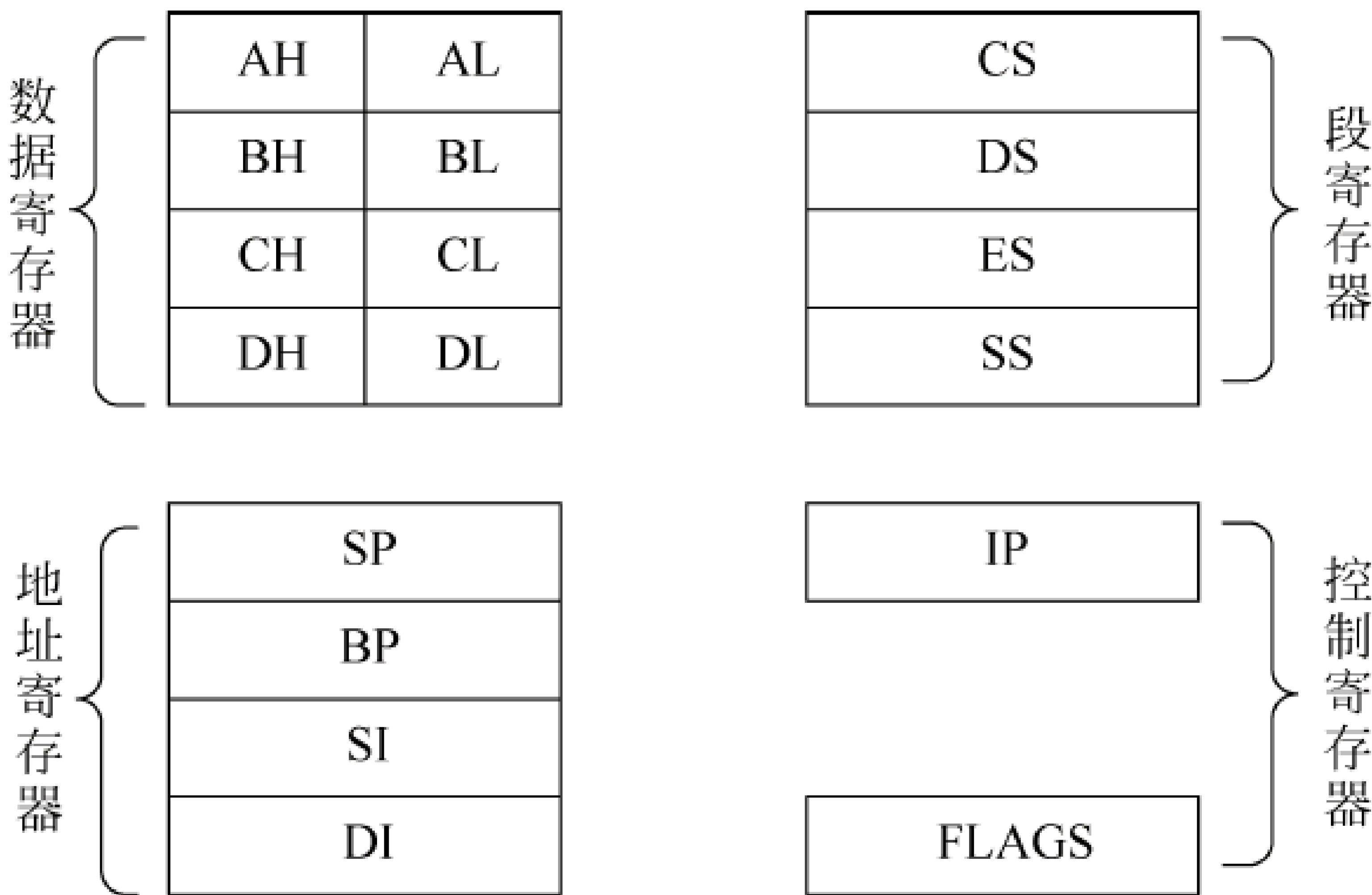


图 2-3 8086 CPU 寄存器组结构

2.2.1　数据寄存器

数据寄存器组包括四个 16 位的寄存器 AX、BX、CX、DX，它们既可以作为 16 位寄存器使用，也可以分为两个独立的 8 位寄存器使用，即高 8 位寄存器 AH、BH、CH、DH 和低 8 位寄存器 AL、BL、CL、DL。这些寄存器既可以作为算术、逻辑运算的源操作数，向 ALU 提供参与运算的原始数据，也可以作为目标操作数，保存运算的中间结果或最后结果。

除了作为通用寄存器以外，在一些指令中，上述寄存器还有一些特殊用法。例如，在算术运算中，一般用 AX/AL 作默认累加器；BX 可以用作地址指针，因而又叫基址寄存器；CX 经常用作循环指令的计数器，又称计数寄存器；DX 作为数据寄存器，与 AX 配合存放 32 位数，其中 DX 存放高 16 位，AX 存放低 16 位。

2.2.2　地址寄存器

地址寄存器分为两个指针寄存器 SP、BP 和两个变址寄存器 SI、DI。这组寄存器通常用来存放存储器单元的 16 位偏移地址。当然，这些寄存器也可以作为通用寄存器使用。

1. 指针寄存器

在进行堆栈操作的过程中，SP 用来指示堆栈栈顶的偏移地址，称为堆栈指针；而 BP 则用来存放位于堆栈段中的一个数据区的“基址”的偏移量，称为基址指针。

2. 变址寄存器

两个变址寄存器 SI、DI，它们用来存放当前数据所在存储单元的偏移地址。在串操作指令中，SI 用来存放源操作数地址的偏移量，称为源变址寄存器；DI 用来存放目标操作数地址的偏移量，称为目标变址寄存器。

2.2.3　段寄存器

在 8086 CPU 中有四个 16 位的段寄存器：CS、DS、ES、SS。这些寄存器指明了一个特定的现行段，用来存放各段的段基址。当用户用指令设定了它们的初值后，实际上已经确定了一个 64KB 的存储区段。

代码段寄存器 CS 用来存放当前使用的代码段的段基址，用户编制的程序必须存放在代码段中，CPU 将会依次从代码段取出指令代码并执行。

数据段寄存器 DS 用来存放当前使用的数据段的段基址，程序运行所需的原始数据以及运算的结果应存放在数据段中。

附加段寄存器 ES 用来存放当前使用的附加段的段基址，它通常也用来存放数据，但在数据串操作时，用来存放目标数据串（此时 DS 用来存放源数据串）。

堆栈段寄存器 SS 用来存放当前使用的堆栈段的段基址，所有堆栈操作的数据均保存在这个段中。

2.2.4　控制寄存器

控制寄存器包括指令指针寄存器 IP 和标志寄存器 FLAGS。

1. 指令指针寄存器 IP

IP 为 16 位指令指针，IP 的内容总是指向 BIU 将要取的下一条指令代码的 16 位偏移

地址,它与 CS 联合确定指令的物理地址。在程序运行过程中,它始终指向下一条指令的首地址。一般情况下,当取出 1B 指令代码后,IP 自动加 1 并指向下一条指令代码的偏移地址。它的内容是由 BIU 来修改的,用户不能通过指令预置或修改 IP 的内容,但有些指令的执行可以修改它的内容,例如转移指令 JMP 会把目标地址的偏移量送入 IP;也可以将其内容压入堆栈或由堆栈中弹出,例如子程序调用指令 CALL 和返回指令 RET 的执行。

2. 标志寄存器 FLAGS

标志寄存器又称程序状态字寄存器(Program Status Word,PSW),它是一个 16 位的寄存器,用来反映微处理器在程序运行时的某些状态。标志寄存器只使用了 9 位,其中 6 位(OF、SF、ZF、AF、PF、CF)为状态标志位,记录算术运算或逻辑运算指令相关的状态信息,另外 3 位(DF、IF、TF)为控制标志,在执行某些指令时起控制作用。8086 CPU 标志寄存器各位的定义如图 2-4 所示。

D15				D11	D10	D9	D8	D7	D6		D4		D2		D0
—	—	—	—	OF	DF	IF	TF	SF	ZF	—	AF	—	PF	—	CF

图 2-4　标志寄存器

有关的标志位含义如下:

- 进位标志 CF(Carry Flag):若最高有效位有进位/借位,则 CF=1;否则,CF=0。
- 符号标志 SF(Sign Flag):运算结果为负时,SF=1;运算结果为非负数时,SF=0。
- 零标志 ZF(Zero Flag):运算结果为 0 时,ZF=1;否则,ZF=0。
- 溢出标志 OF(Overflow Flag):当运算结果超出了机器所能表达的数的范围时称为溢出,此时 OF=1;否则,OF=0。一般情况下,同号数相加或异号数相减可能会产生溢出。
- 辅助进位标志 AF(Auxiliary carry Flag):当低位字节中的 D_3 位向 D_4 位有进位/借位时,AF=1;否则,AF=0。
- 奇偶标志 PF(Parity Flag):若操作结果低 8 位中含 1 的个数为偶数,则 PF 置 1,否则置 0。
- 方向标志 DF(Direction Flag):用来设定和控制字符串操作指令的步进方向。DF=1 时,串操作过程中的地址会自动递减 1;DF=0 时,地址自动递增 1。
- 中断标志 IF(Interrupt Flag):用来控制可屏蔽中断的标志位。IF=1 时,开中断,CPU 可以接收可屏蔽中断请求;IF=0 时,关中断,CPU 不能接收可屏蔽中断请求。
- 陷阱标志 TF(Trap Flag):用来控制 CPU 进入单步工作方式。TF=1 时,8086 CPU 处于单步工作方式,每执行完一条指令就自动产生一次内部中断;TF=0 时,CPU 不能以单步方式工作。

2.3　存　储　器

2.3.1　存储单元的地址和内容

在计算机中,数据的最小存储单位是一个二进制位(b),一位可存储一个二进制数 0 或

1。把 8 个连续的二进制位组合在一起就构成一个字节(B)。8086 CPU 有 20 根地址线,可直接寻址 1MB 的存储空间。为了便于对存储器进行存取操作,每一个存储单元都有一个唯一的地址与之对应,其地址范围用十六进制表示为 00000H～FFFFFH。

在汇编语言中,把存储单元分为字节单元、字单元、双字单元等,称为存储单元的属性。存储器以字节为编程单位,一个字节单元可以存储 8 位二进制数。一个字单元要占用相继的两个字节单元,可以存储 16 位二进制数,低位字节存入低地址,高位字节存入高地址,字单元地址用它的低地址来表示。双字单元要占用相继的四个字节单元,可以存储 32 位二进制数。

在存储单元中的数据称为存储单元内容。例如,将字节数据 35H 存放在 136E1H 单元,字数据 8A9DH 存放在 136E2H 单元,存储单元的地址和内容,如图 2-5 所示。

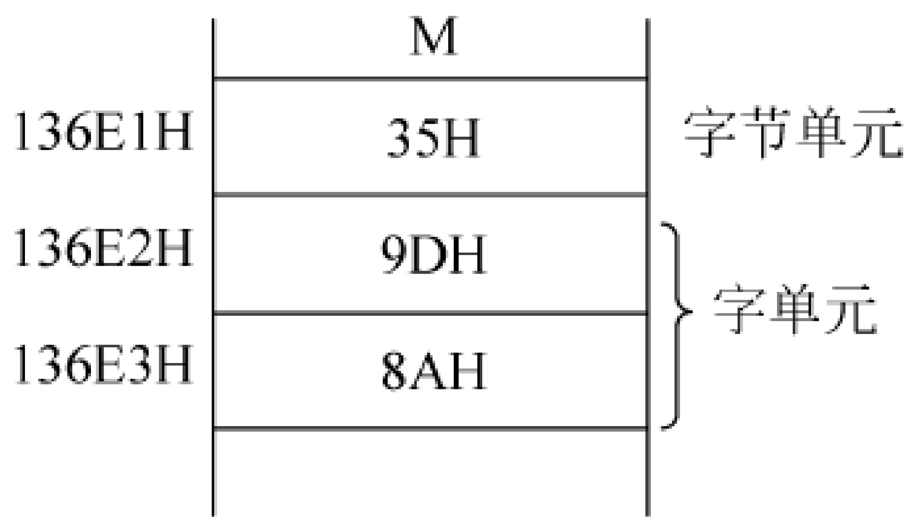


图 2-5　存储情况示意图

由于 8086 CPU 是以偶地址访问(读/写)存储器,因此存储字数据时,一般放在偶地址单元中。如果在奇地址单元存储一个字,则需要两次访问存储器才能读出一个字。

为便于理解和区分,表 2-1 列出了寄存器与存储器的特点对比,这对于学习汇编语言和理解微型计算机组织结构尤为重要。

表 2-1　寄存器与存储器的特点对比

寄　存　器	存　储　器
在 CPU 内部	在 CPU 外部
访问速度快	访问速度慢
容量小,成本高	容量大,成本低
用名字表示	用地址表示
没有地址	地址可用各种方式形成

2.3.2　存储器分段

8086 CPU 有 20 根地址总线,可以访问存储器的最大容量为 1MB。由于所有可用来存放地址的寄存器都是 16 位的,无法直接提供 20 位地址,因此采用了存储器地址分段的办法。把 1MB 的存储空间划分成若干个段(Segment),每个段可由 1～64KB(即 65 536B)个连续的字节单元组成。每个段是一个可独立寻址的逻辑单位。在 8086 汇编语言的程序设计中,需要设立几个段,每个段有多少个字节以及每个段的用途完全由用户自己确定。同时每个段中存储的代码或数据,可以存放在段内任意单元中。

在任意时刻,程序能够很方便地访问四个分段的内容。这四个分段分别是代码段、数据段、堆栈段和附加段。将这四个段的起始地址的最高 16 位地址值(用十六进制表示为四位)分别存放在 CS、DS、SS 和 ES 四个段寄存器中,称为段基址,由这四个段寄存器指向的段称为当前段。利用指令可以任意设定段寄存器的内容,段基址一旦确定,对应 64KB 的存储区段则完全确定下来,程序可以从四个段寄存器给出的逻辑段中存取指令代码和数据。在物理上,这四个段可以在内存的任何地方,覆盖或不覆盖,唯一要求是各段起始地址能被 16 整除。

2.3.3 逻辑地址与物理地址

每个存储单元有两种形式的地址：逻辑地址和物理地址。在汇编语言程序设计中，用户编程使用的是逻辑地址，而不关心物理地址。一个逻辑地址是由段基值和偏移地址(偏移量)组成。段基值放在段寄存器中，是一个段首地址的高16位；偏移地址是某存储单元与其对应的段首地址之间的字节距离。

8086地址总线20位，最大寻址空间1MB，而8086字长为16位，这就需要解决16位字长的机器如何提供20位地址的问题。CPU在访问存储器时，用16位的地址指针指向存储器，需要把逻辑地址转换为物理地址。转换方法是采用“双部”寻址方案，即16位地址指针只是物理地址的一部分，它要与16位段寄存器的内容一起联合形成20位的物理地址。它很好地解决了16位地址指针和20位物理地址之间的矛盾。

物理地址的形成，通常用下面的公式来表达：

20位的物理地址 ＝16位段寄存器内容左移4位＋16位地址指针

或

20位的物理地址＝16位段寄存器内容＊10H＋16位地址指针

8086存储器物理地址的形成过程如图2-6所示。

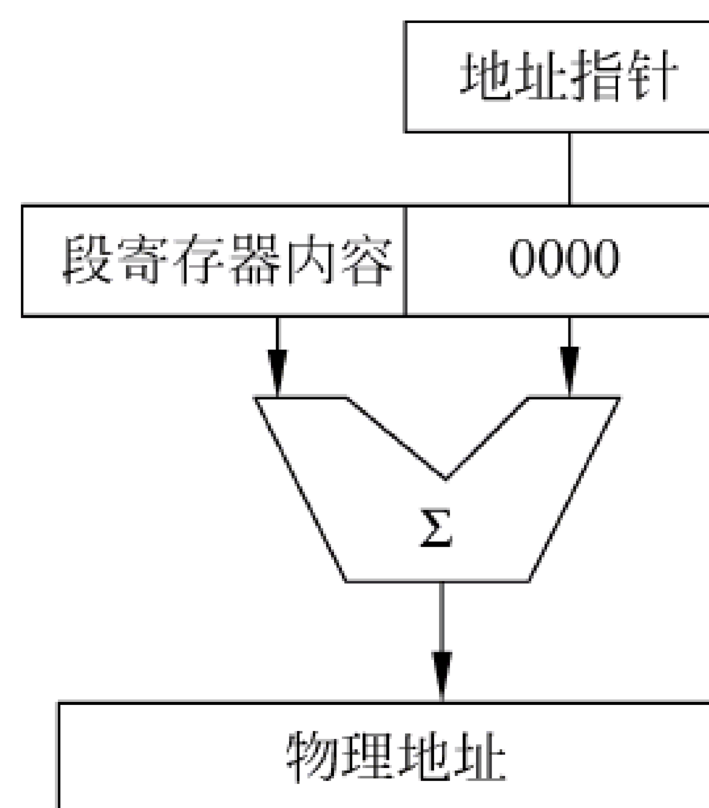


图2-6 物理地址的形成过程

例如，数据段DS＝136EH，偏移地址为0100H的数据段的存储单元，其物理地址为137E0H。

2.3.4 堆栈

堆栈(Stack)是内存中一个特殊的存储区域，是一端固定一端活动的存储空间。固定端称为栈底，活动端称为栈顶。数据的存取方式为后进先出(Last In First Out，LIFO)，就像子弹壳装弹，一粒一粒压进去，但是打出来的时候是从上面打出来的，最先压进去的最后弹出来，如果进去顺序是abc，打出来顺序则是cba。堆栈中数据的存取是靠堆栈指针(SP)来完成的，SP总是指向栈顶。随着堆栈中数据的变化，SP值也做相应的变化，例如将一个字的内容压入堆栈，SP的内容会减2；如果将栈顶一个字的内容弹出，则SP的内容会加2。堆栈的主要用途是现场数据保护、子程序与中断程序的调用返回等。

2.3.5 存储器访问

8086在存储器中存储的信息包括程序指令、数据及计算机运行的状态等。为了便于寻址和操作，这些信息在存储器中分段存储，因而将存储器划分为程序区、数据区和堆栈区，并通过段寄存器CS、DS、ES和SS进行寻址。

(1) 取指令：系统自动选择CS，再加上IP确定的16位偏移量，计算得到要取的指令的实际物理地址。

(2) 堆栈操作：自动选择SS，偏移取SP或BP值。

(3) 涉及操作数时：自动选择DS或ES，再加上16位偏移量，计算得到20位的物理地址。16位偏移量的值，取决于指令的寻址方式。

(4) 在不改变段寄存器值的情况下，最大寻址范围是64KB。

2.4　外部设备

外部设备与主机的通信是通过外设接口(Interface)进行的,每个接口包括一组寄存器。外设中每个寄存器有一个端口(Port)地址,构成一个独立于内存的 I/O 地址空间。由于 8086 用地址总线的低 16 位 A15～A0 来寻址端口地址,因此 8086 CPU 可以访问的 I/O 端口地址共有 64KB,其地址为 0000H～FFFFH。这些端口均为 8 位端口(即通过该端口一次输入输出一个字节信息)。对端口的寻址有直接寻址方式和间接寻址方式两种。直接寻址适用于地址为 00H～FFH 的端口寻址。间接寻址适用于地址为 0100H～FFFFH 的端口寻址(所有端口均可采用间接寻址方式)。

为便于用户使用外部设备,系统提供了两种类型的功能调用,一种是 BIOS(Basic Input Output System)功能调用,另一种是 DOS(Disk Operating System)功能调用。它们都是系统编制的子程序,通过中断方式转入所需要的子程序去执行,执行完后返回原来的程序继续执行。操作系统把一些复杂的外设操作编成例行程序,使用户通过中断指令(INT)就可以进入这些例行程序,完成所需要的外设操作。所以,用户应尽量利用这些系统所提供的工具来编写自己的程序。

BIOS 和 DOS 功能调用虽然都是系统提供的例行程序,但是它们之间又有差别。BIOS 存放在机器的只读存储器 ROM 中,所以可以把它看成是机器硬件的一个组成部分,它的层次比 DOS 功能调用更低。DOS 功能调用,是在开机时由磁盘装入存储器,在它的例行程序中可以一次或多次调用 BIOS,以完成比 BIOS 更高级的功能。用户需要使用外设时,应尽可能使用层次较高的 DOS 功能调用,但有时它不能满足应用需求,就需要直接调用 BIOS,如果 BIOS 还不能解决问题,那么就只能自己编制中断处理程序了。

2.5　通过 DEBUG 使用存储器和寄存器

在 1.3 节中,已经介绍了 DEBUG 的初步使用方法,这里再结合有关存储器和寄存器的使用,进一步了解 DEBUG 的常用命令。

1. 显示存储单元命令 D

格式:

```
D [地址]        ;显示当前或指定开始地址的主存内容
D [范围]        ;显示指定范围的主存内容
```

使用时,[地址]和[范围]是可选项,所以会出现多种形式。

【例 2-1】 只写一个 D,表示显示从当前地址单元开始的 128 个单元中的字节数据。

```
-D
136E:0100  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0110  00 00 00 00 00 00 00 00-00 00 00 00 34 00 5D 13   ............4.].
136E:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

```
136E:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

显示的区域分为三部分。最左边的为地址列表区,列出每行 16 个数据的起始地址,同行中的每个单元的地址采用类推得到。中间区域为数据列表区,每行 16 个数据,8 行共 128 个数据。在一行中,前 8 个数据和后 8 个数据用分隔符“-”隔开。第三部分为可显示字符区,用来表示一行的 16 个数据中,哪些是 ASCII 码表中的可显示字符。如果是可显示字符,则显示出该字符,如果不是可显示字符,则用黑点“.”表示。所有的数据都默认为 16 进制。

从显示的内容,可得到如下信息:

(1) 当前单元所在段的段地址为 136EH,当前单元的偏移地址为 0100H。所以第一行第一个单元地址为 136EH:0100H,其中存放的内容为 00H。第一行第二个数据即是下一个单元的内容,依次类推。

(2) 在第二行的 16 个数据中,011CH 单元是 34H,在 ASCII 码表中表示“4”字符。所以可显示字符区对应位置是 4。

【例 2-2】 指定要显示的单元的逻辑地址,即写出完整的段地址和偏移地址,DEBUG 会显示从指定单元开始的 128 个单元的字节数据。

```
-D  136E:011B
136E:0110                                   00 34 00 5D 13              .4.].
136E:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0180  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0190  00 00 00 00 00 00 00 00-00 00 00                  ...........
-
```

【例 2-3】 程序运行时,只有一个当前数据段,所以可以用 DS 代替段地址的具体值,或者省略段地址部分,默认访问当前数据段,只需指明偏移地址即可。

```
-D DS:0110 0118
136E:0110  00 00 00 00 00 00 00 00-00                        .........
-
```

【例 2-4】 显示多个单元内容时,也可以通过指定起始单元地址后,通过范围表明显示单元的个数。范围以 L 开始,后面为十六进制表示的单元个数。

```
-D 0110 L 8
136E:0110  00 00 00 00 00 00 00 00                           ........
-
```

2. 修改存储单元命令 E

E 命令用于修改主存单元内容,有以下两种格式。

格式：

```
E地址 数据表      ;用数据表的数据修改指定单元的内容
E地址             ;查看指定单元内容后再修改
```

格式 1 可以使用数据表一次修改一个或多个单元的内容。

数据表中为要写入每个单元的数据，可以是十六进制数据，也可以是单引号括起来的字符或字符串。数据和数据间要用空格间隔，数据和字符可以不分隔。注意，E 命令完成修改后，屏幕上并不会显示执行结果，需要用 D 命令查看修改后的结果。

【例 2-5】 用数据表形式修改指定起始单元开始的多个单元值。

前面我们用 D 命令查看到 136E:0100 单元的值为 00H，136E:0101 单元的值为 00H。现在用 E 命令将 0100H 单元的值修改为 01H，0101H 单元的值修改为 02H。

```
- E 136E:0100 01 02
```

要确认是否已经修改成功，用 D 命令查看：

```
-D 136E:0100 0101
136E:0100   01 02                                                  ..
```

【例 2-6】 将字符或字符串存放到指定存储单元，可以用字符的 ASCII 码形式，也可以为单引号括起来的字符形式。

现在我们将 0100H 和 0101H 单元修改为字符'A'，'a'(注意大小写字母的 ASCII 码是不同的），一个采用 ASCII 码形式，一个采用字符直接输入形式。

```
-E DS:0100 41 'a'
```

用 D 命令可以看到单元的值已经修改成功。

```
-D DS:0100 0101
136E:0100   41 61                                                  Aa
-
```

【例 2-7】 字符串存放到存储单元中，可以依次放入每个字符的 ASCII 码，也可以是用单引号括起来的多个字符形式。

将字符串'ABCD'存放到 0100H 单元开始的连续 4 个单元中。

```
-E 0100 'ABCD'
```

或者

```
-E 0100 41 42 43 44
-E 0100 'ABCD'
-D
136E:0100   41 42 43 44 00 00 00 00-00 00 00 00 00 00 00 00   ABCD............
136E:0110   00 00 00 00 00 00 00 00-00 00 00 00 34 00 5D 13   ............4.].
136E:0120   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0130   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0140   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0150   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0160   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

```
136E:0170   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

3. 填充命令 F

F 命令用于对一个主存区域填写内容,这样会改写原来的内容。

格式:

```
F 地址 范围 数据表
```

该命令将数据表的数据写入从指定起始地址开始的一定范围的主存区域中。如果数据个数超过指定的单元个数范围,则忽略多出的数据项;如果数据个数小于指定的单元个数范围,则重复使用这些数据,直到填满指定的范围区域。

【例 2-8】 我们将 136E:0100 开始的 16 个单元全部填充为数据十进制的 15。因为 DEBUG 中默认进制为十六进制,所以数据表应为 0FH。16 个单元数用十六进制表示为 10H,注意个数前面要加上 L。F 命令的执行结果,也是用 D 命令查看。

```
-F 136E:0100 L10 0F
-D 136E:0100 010F
136E:0100   0F 0F 0F 0F 0F 0F 0F 0F-0F 0F 0F 0F 0F 0F 0F 0F   ................
-
```

【例 2-9】 将 136E:0100 单元开始的 16 个单元全部用字符串'ABCD'填充。数据表中用字符串或者每个字符的 ASCII 码都可以,注意 ASCII 码要用空格分隔。

```
-F 136E:0100 L10 'ABCD'
```

或者

```
-F 136E:0100 L10 41 42 43 44
-D 136E:0100 010F
136E:0100   41 42 43 44 41 42 43 44-41 42 43 44 41 42 43 44   ABCDABCDABCDABCD
-
```

4. 寄存器显示或修改命令 R

R 命令用于显示和修改处理器中的寄存器,它有三种格式。

格式:

```
R               ;显示 CPU 内所有寄存器内容和标志寄存器中标志位值
R 寄存器名       ;显示和修改指定的寄存器
RF              ;显示和修改标志寄存器中的标志位
```

【例 2-10】 查看当前 CPU 中各寄存器的值。

```
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0100   NV UP EI PL NZ NA PO NC
136E:0100 41           INC     CX
-
```

从显示结果可以看出,当前 CPU 中各寄存器的值(通用寄存器、标志寄存器、段寄存器)。并且,由寄存器中值的特殊含义,可以知道主存的情况。上面的显示信息表明,现在的

内存区域内，数据段(DS)、代码段(CS)、堆栈段(SS)、附加段(ES)的值都为 136EH。

有些寄存器中的值是有其特定含义的，例如 IP 表明下一条要执行的指令的偏移地址，SP 表明堆栈区中现在栈顶单元的偏移地址是 FFEEH。可以在第三行看到下一条要执行的指令的汇编代码、机器代码，以及指令存放的起始地址，因为指令转换为机器码后长度可以由 1～7 个字节组成，指令不同长度可能不同，所以只给出指令的起始地址。另外标志寄存器的值用符号表示，每个符号代表的状态含义不同。表 2-2 给出了每个状态位的中英文含义。

表 2-2　标志位的符号表示

标　志　名		标志为 1	标志为 0
OF	溢出(是/否)	OV (OVerflow)	NV (No oVerflow)
DF	方向(减量/增量)	DN (DownN)	UP (UP)
IF	中断(允许/关闭)	EI (Enable Interrupt)	DI (Disable Interrupt)
SF	符号(负/正)	NG (NeGative)	PL (Plus)
ZF	零(是/否)	ZR (ZeRo)	NZ (No Zero)
AF	辅助进位(是/否)	AC (Auxiliary Carry)	NA (No Auxiliary)
PF	奇偶(偶/奇)	PE (Parity Even)	PO (Parity Odd)
CF	进位(是/否)	CY (Carry Yes)	NC (No Carry)

【例 2-11】　要修改寄存器的值，则 R 命令中就需要指明要修改的寄存器名。例如，要将 AX 中的值修改为 1234H：

```
- R AX
```

屏幕上显示当前 AX 的值为 0000H，冒号后等待用户输入新数据，如果需要修改就输入新的数据(字符用 ASCII 码)，如果不修改则按回车键结束命令。

```
-R AX
AX 0000
:1234
-R
AX = 1234   BX = 0000   CX = 0000   DX = 0000   SP = FFEE   BP = 0000   SI = 0000   DI = 0000
DS = 136E   ES = 136E   SS = 136E   CS = 136E   IP = 0100    NV UP EI PL NZ NA PO NC
136E:0100 41              INC     CX
-
```

【例 2-12】　修改标志寄存器的值，用 RF 命令。

```
- RF
```

屏幕上会显示当前标志寄存器的标志位情况，在"-"后，输入要修改的标志位的符号表示即可。不修改或修改完成按回车键。现在要让 CF＝1，DF＝1，ZF＝1，则需要输入 CY，DN，ZR。输入的顺序可以任意。用 R 命令可以查看修改后的情况。

```
-RF
NV UP EI PL NZ NA PO NC    -CY DN ZR
-R
AX = 1234   BX = 0000   CX = 0000   DX = 0000   SP = FFEE   BP = 0000   SI = 0000   DI = 0000
DS = 136E   ES = 136E   SS = 136E   CS = 136E   IP = 0100    NV DN EI PL ZR NA PO CY
136E:0100 41              INC     CX
```

2.6 实验内容

【实验目的】 通过 DEBUG 常用命令的使用，特别是 D、E、F、R 等命令的运用，进一步理解存储器和寄存器的功能和使用。

【实验 2-1】 在 DEBUG 下，完成下列操作，说明操作方法并记录运行结果。

(1) 设置寄存器的初值：AX＝1A2BH，BX＝3456H，DL＝78H，DH＝90H，DS＝2000H，SS＝3000H。

(2) 查看各寄存器初值设置情况，然后修改寄存器内容，使 AX＝1234H，BX＝5678H。

(3) 将 DS 指定的数据段 0100H 开始的 10 个单元，用字母'abcde'填充；后继的 10 个单元分别存放内容是：'A'，'B'，'C'，'D'，'E'，41H，42H，43H，44H，45H。

(4) 查看 DS 数据段 0100H 开始的存储区域数据。

(5) 将(2000H:0101H)单元内容修改为 66H，(2000H:0102H)单元内容修改为 67H。

(6) 将下列数据存放在数据段(136EH:0100H)单元开始的存储区域中。

(136EH:0100H)＝28H

(136EH:0101H)＝'A'

(136EH:0104H)＝3219H

(136EH:0106H)＝1234ABCDH。

(7) 查看(136EH:0100H)单元开始的存储区数据存放情况。

(8) 设置标志寄存器，使其达到运算结果为零并有进位的状态。

习　　题

1. 8086 CPU 寄存器有哪些？各有什么功能和用途？哪些寄存器既可以当 16 位寄存器使用，又可以分成两个独立的 8 位寄存器？

2. 标志寄存器中与运算结果有关的状态信息是哪几位？各是什么含义？

3. 如何解决 16 位字长的机器与 20 位物理地址之间的矛盾？若在 DEBUG 中显示的地址是(136E:0200)单元，则其物理地址是什么？

4. 将数据 1020H 存入 13670H 单元、数据 2DH 存入 13672H 单元、数据 89ABCDEFH 存入 13674H 单元后，数据如何存放？请画出示意图。如何在 DEBUG 环境下验证？请说明操作步骤。

5. 存储器分段时，每个逻辑段最大是多少个字节单元？设数据段的段基值为 136EH，写出该段的首单元和末单元的物理地址。

第3章

汇编指令与寻址方式

全面、准确地理解和掌握每条汇编指令的功能和用法，是利用指令编写汇编语言程序的关键。在学习每条指令的时候，应该关注指令的功能、寻址方式、是否影响标志位等。本章主要介绍8086 CPU指令系统及常用的寻址方式，重点是与数据有关的寻址方式，并结合DEBUG上机验证。

3.1 指令和指令系统

3.1.1 汇编指令

1. 指令

通常一条汇编指令对应着一条机器指令，它是计算机能够直接识别并执行的二进制代码，代表一种基本操作，例如：加法、减法、乘法、除法等。计算机中的指令由操作码和操作数两部分组成。操作码说明计算机要执行的操作，而操作数是在指令执行过程中所需要的操作对象。操作数部分可以是操作数本身，也可以是操作数地址或是与操作数有关的其他信息。计算机所能执行的全部指令构成了计算机的指令系统。每种计算机都有各自的指令系统，80x86系列的指令系统是向上兼容的。

2. 代码指令的格式

当一条汇编指令译成代码指令时，由汇编指令中的操作码和操作数共同决定代码指令的格式。对8086 CPU，代码指令占1～6B不等。第一个字节固定为操作码(OPCODE)，其余字节为操作数，操作数可能有0个、1个、2个或3个，操作数字节的确定与指令的寻址方式有关。代码指令的一般格式如图3-1所示。

操作码	操作数	…	操作数

图3-1 代码指令的一般格式

在汇编语言中，用助记符表示操作码，用符号或符号地址表示操作数或操作数所在的地址，与机器指令一一对应。8086 CPU的指令系统中，主要包括数据传送指令、算术指令、逻辑指令、串操作指令、控制转移指令、处理机控制与杂项操作指令。为了使学生能够循序渐进地学习汇编指令，本书把指令系统分散到后续各章介绍，使指令系统的学习与汇编语言程序设计有效地结合，避免过多地学习指令系统，令学生感到厌倦。

学习指令系统时，应重点关注指令的汇编格式、指令的基本功能、指令支持的寻址方式、指令的执行对标志位的影响、指令的特殊要求等，切忌死记硬背。因此，本节不再详细介绍

具体指令的使用。

3.1.2 汇编指令的书写形式

1. 格式

```
[name]  operation  operand  [;comment]
```

名字项(name)：可以是标号或变量，表示本语句的符号地址。标号在代码段中定义，后跟冒号。变量在数据段或附加段中定义，后面不跟冒号。

名字项组成：以字母打头的字符串组成，包含字符(A～Z，0～9，_，?，$，@等)，如MainLoop，Calc_long_sum。单独的$或"?"有特殊含义，不能做符号名。保留字不能用在名字项。

操作项(operation)：可以是指令、伪指令、宏指令的助记符。

操作数项(operand)：可以是一个或多个表达式组成，多个操作数之间用逗号分隔。操作数可能是一个或多个，也可能没有。表达式中涉及的操作符如下：

- 算术操作符：+，-，*，/，MOD；
- 逻辑操作符：AND，OR，XOR，NOT，SHL，SHR；
- 关系操作符：EQ，NE，LT，GT，LE，GE；
- 其他操作符：TYPE，LENGTH，SIZE，OFFSET，SEG，PTR，THIS等。

注释项(comment)：可选项，用来说明程序或指令的功能。一般由分号";"开始直至回车键结束均视为注释部分。

2. 操作数的主要类型

立即操作数：也称立即数，可采用二、八、十、十六进制，后缀分别为B、Q或O、D、H，若十六进制以字母开头，必须加前导"0"。立即数是作为指令代码的一部分出现在指令中，它通常作为源操作数使用，给寄存器、地址赋初值、循环数等。

寄存器操作数：寄存器的内容参加运算或存放结果。寄存器操作数是把操作数存放在CPU中的寄存器内，即用寄存器存放源或目的操作数。在汇编指令中给出寄存器的名称。在双操作数指令中，可以做源也可以做目的操作数。

存储器操作数：指内存某地址的字节、字、双字等是指令的处理对象，这时必须把处理对象取出或送入相应地址。存储器操作数所在的存储器地址应该是物理地址，但在汇编指令中，通常只给出有效地址EA(它是以各种寻址方式给出的)，而段基值(在段寄存器中)是通过隐含方式(或段超越)使用的。

3. 有效地址和段超越

存储单元的物理地址由两部分组成：段寄存器保存的段基值和偏移地址。在8086的寻址方式中，存储单元所需的偏移地址，称为有效地址，用EA表示。不同的寻址方式，组成有效地址EA的各部分内容也不一样，寻址方式主要是EA如何计算与寻找的问题。

一般情况下，使用寻址方式中规定的默认段寄存器来确定段的基地址。指令中的操作数若不在原默认的区段内，必须在指令中指定段寄存器，这称为段超越。一般操作数可以存放在代码段、数据段、堆栈段或附加段中，而指令必须在代码段中，堆栈也只能在堆栈段中。段地址的基本规定以及允许段超越的情况如表3-1所示。

表 3-1　段地址的基本约定以及允许的段超越情况

访问存储器的方式	默　认　段	允许段超越	偏 移 地 址
取指令	CS	无	IP
堆栈操作	SS	无	SP
一般数据访问	DS	CS,ES,SS	有效地址 EA
BP 作为基址的寻址	SS	CS,ES,DS	BP
串操作的源操作数	DS	CS,ES,SS	SI
串操作的目的操作数	ES	无	DI

3.2　寻 址 方 式

寻址方式是汇编语言中非常重要的内容,只有掌握好各种寻址方式的使用,才能真正理解汇编语言的编程方法。寻址方式有两大类,一类是与数据有关的寻址方式,另一类是与转移地址有关的寻址方式。这里介绍的基本的寻址方式都是与数据有关的,简单地说寻址方式就是寻找操作数的方法。操作数作为操作对象,有时直接给出,也有时不直接给出,而是给出操作数的有效地址或者给出得到操作数有效地址的计算方法。如何得到操作数,这就是寻址方式的主要内容。

本节重点介绍与数据有关的 8086 寻址方式,为便于理解寻址方式的实例,这里先给出 MOV 指令的一般用法。

格式:MOV 目的操作数,源操作数

功能:(目的操作数)←(源操作数),即数据从源操作数的副本传送到目的操作数中,它类似于复制操作。

3.2.1　立即寻址方式

在立即寻址方式中,操作数在指令中直接给出,紧跟在操作码的后面存储在代码段中。立即数可以是 8 位或 16 位,对于 16 位立即数,存储时仍然遵循“低位在前,高位在后”的原则。

指令格式:MOV　目的操作数,立即数

例如:

```
MOV  AL,5                 ;(AL)←5
MOV  AH,0FFH              ;(AH)←FFH
MOV  AX,3064H             ;(AX)←3064H
MOV  AL,'A'               ;(AL)←字符'A'的 ASCII 码
```

说明:

(1) 立即数存储在代码段中。

(2) 立即数只能用于源操作数字段,如“MOV　40H, AL”错误。

(3) “源”和“目的”的字长一致,如“MOV　AH, 3064H”错误。

【例 3-1】　指令“MOV AX,3864H”执行后,AX=?

指令执行后,立即数 3864H 存入寄存器 AX,操作示意图如图 3-2 所示。



图 3-2 立即寻址操作示意图

在 DEBUG 中，先用 A 命令输入汇编指令，然后用 T 命令单步执行这条指令，指令中的源操作数 3864H 就传送到了 AX 寄存器中。执行结果如下：

```
-A
136E:0100 MOV AX,3864
136E:0103
-T
AX=3864  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0103   NV UP EI PL NZ NA PO NC
136E:0103 0000          ADD     [BX+SI],AL                          DS:0000=CD
-
```

3.2.2 寄存器寻址方式

寄存器寻址方式是指操作数存放在处理器内部的寄存器中。在汇编语言中，直接利用寄存器名来表达操作数所在的寄存器。操作数可以是 8 位或 16 位寄存器，但要注意字长的匹配。

说明：

(1) 字节寄存器只有：AH，AL，BH，BL，CH，CL，DH，DL。

(2) 16 位寄存器：AX，BX，CX，DX，SI，DI，SP，BP。

(3) “源”和“目的”的字长一致，如“MOV AH, BX”错误。

(4) CS 不能用 MOV 指令改变，如“MOV CS, AX”错误。

【例 3-2】 若指令执行前，AX＝3864H，BX＝1234H，则执行下列两条指令后，寄存器内容如何变化？

```
MOV   AX, BX              ;寄存器 BX 的内容传送给 AX,AX=1234H
MOV   AL, BH              ;寄存器 BH 的内容传送给 AL,AL=12H
```

在 DEBUG 中验证结果如下：

```
-R AX
AX 0000
:3864
-R BX
BX 0000
:1234
-R
AX=3864  BX=1234  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0100   NV UP EI PL NZ NA PO NC
```

```
136E:0100 0000          ADD     [BX+SI],AL                          DS:1234=00
-A
136E:0100 MOV AX,BX
136E:0102 MOV AL,BH
136E:0104
-T
AX=1234  BX=1234  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0102   NV UP EI PL NZ NA PO NC
136E:0102 88F8              MOV      AL,BH
-T
AX=1212  BX=1234  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0104   NV UP EI PL NZ NA PO NC
136E:0104 0000          ADD     [BX+SI],AL                          DS:1234=00
-
```

【例3-3】 说明下列指令采用的寻址方式和执行结果：

```
MOV  AX,0ABCDH
MOV  BX,AX
MOV  AH,AL
(AX)=?  (BX)=?
```

解析：第一条指令，采用立即寻址，将立即数0ABCDH存入寄存器AX，则(AX)=0ABCDH；第二条指令，采用寄存器寻址，执行后，(BX)=0ABCDH，AX内容不变；第三条指令，采用寄存器寻址，执行后，(AH)=0CDH，(AL)=0CDH。最终结果，(AX)=0CDCDH，(BX)=0ABCDH。

DEBUG下验证结果：

```
-A
136E:0100 MOV AX,ABCD
136E:0103 MOV BX,AX
136E:0105 MOV AH,AL
136E:0107
-T
AX=ABCD  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0103   NV UP EI PL NZ NA PO NC
136E:0103 89C3              MOV      BX,AX
-T
AX=ABCD  BX=ABCD  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0105   NV UP EI PL NZ NA PO NC
136E:0105 88C4              MOV      AH,AL
-T
AX=CDCD  BX=ABCD  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0107   NV UP EI PL NZ NA PO NC
136E:0107 0000          ADD     [BX+SI],AL                          DS:ABCD=00
-
```

3.2.3　直接寻址方式

在直接寻址方式的指令中直接包含了操作数的有效地址，即EA由指令直接给出。EA

就在指令的操作码后面的操作数字段中，且按 EA 的低字节在前，高字节在后的形式存放。直接寻址方式直接给出了操作数在主存中的偏移地址，实际的物理地址应由段基值与这个直接给出的有效地址 EA 的组合来决定。一般形式为[偏移地址]，或者为符号变量，但在 DEBUG 中不能使用符号变量这种形式。

说明：

(1) 隐含的段为数据段 DS，有效地址 EA 存储在代码段中。

(2) 可使用段跨越前缀，如“MOV　AX，ES：[2000H]”。

【例 3-4】 说明“MOV AX，[2000H]”指令的执行情况。

在指令中给出了 EA＝2000H，假设(DS)＝3000H，那么物理地址 PA＝32000H。在该地址单元中的内容是操作数，如图 3-3 所示。执行结果为(AX)＝3050H。

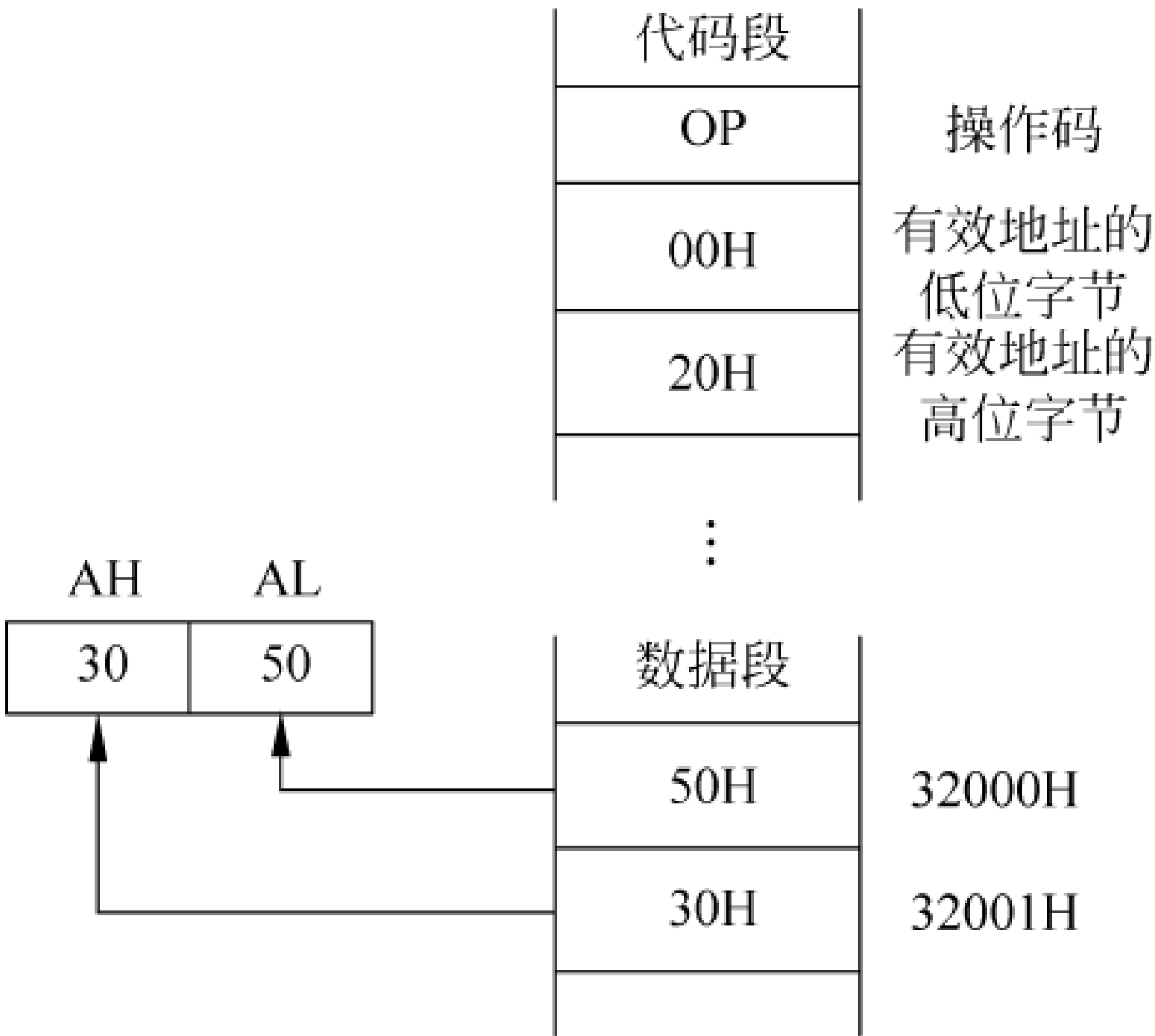


图 3-3　直接寻址示意图

在 DEBUG 中验证时，先用 R 命令设置初始值(DS)＝3000H，再用 E 命令设置(3000H：2000H)单元内容为 50H，(3000H：2001H)单元内容为 30H，然后执行汇编指令 A，执行指令后，可以看到(AX)的内容为 3050H。运行结果如下：

```
-R DS
DS 136E
:3000
-E DS:2000 50 30
-D DS:2000 2001
3000:2000   50 30                                              P0
-A
136E:0100 MOV AX,[2000]
136E:0103
-T
AX = 3050   BX = 0000   CX = 0000   DX = 0000   SP = FFEE   BP = 0000   SI = 0000   DI = 0000
DS = 3000   ES = 136E   SS = 136E   CS = 136E   IP = 0103    NV UP EI PL NZ NA PO NC
136E:0103 0000              ADD      [BX + SI],AL                              DS:0000 = 00
-
```

3.2.4　寄存器间接寻址方式

寄存器间接寻址方式：有效地址 EA 存放在基址寄存器(BX/BP)或变址寄存器

(SI/DI)中，再根据有效地址访问到存储单元中的操作数。它与寄存器寻址不同，这种寻址方式中，寄存器的内容是操作数的有效地址。

说明：

(1) 有效地址 EA 只能放在 SI、DI、BX、BP，不允许使用其他寄存器。

(2) “源”和“目的”的字长要匹配，例如：

```
MOV   DL, [ BX ]          ;[BX]指示一个字节单元
MOV   DX, [ BX ]          ;[BX]指示一个字单元
```

(3) 适于数组、字符串、表格的处理。

寄存器间接寻址的地址形成，可概括为表 3-2。

表 3-2 寄存器间接寻址的地址形成

变 址 器	书 写 形 式	EA	默认段寄存器	物 理 地 址
SI	[SI]	(SI)	DS	(DS) * 10H+(SI)
DI	[DI]	(DI)	DS	(DS) * 10H+(DI)
BX	[BX]	(BX)	DS	(DS) * 10H+(BX)
BP	[BP]	(BP)	SS	(SS) * 10H+(BP)

【例 3-5】 说明“MOV　AX,[BX]”指令的执行情况。

若(DS)=2000H,(BX)=1000H,存储器中数据如图 3-4 所示。寄存器 BX 的内容不是操作数，而是操作数在存储器中的偏移地址，即有效地址 EA=1000H,物理地址为 DS 寄存器内容左移四位再加上 EA,即 PA=21000H。执行时，从低地址单元 21000H 中取出的内容 A0H 送寄存器 AL,从高地址单元 21001H 中取出的内容 50H 送寄存器 AH。执行结果，寄存器(AX)=50A0H。

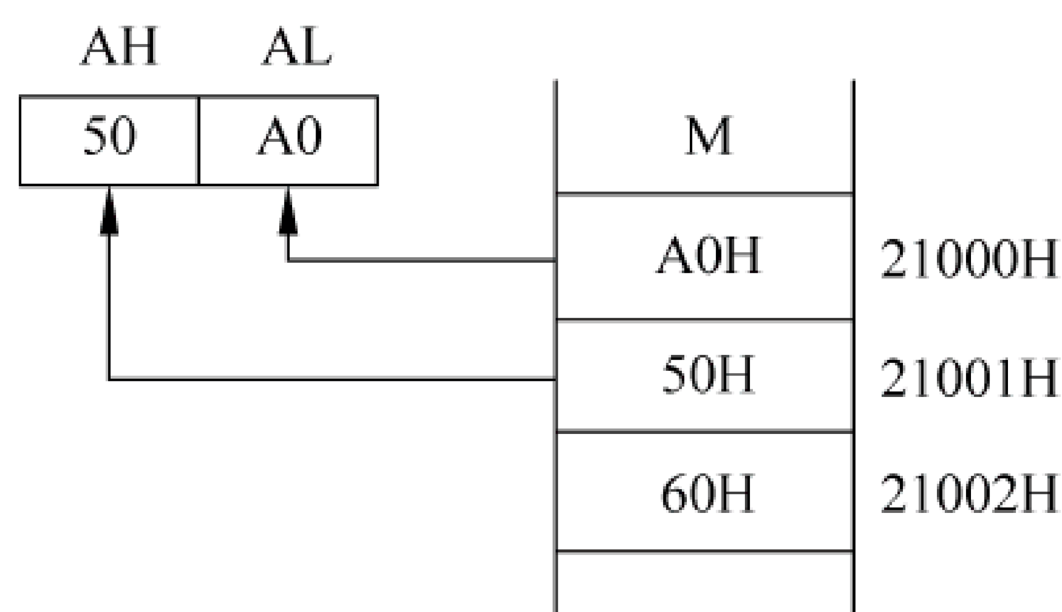


图 3-4 寄存器间接寻址示意图

在 DEBUG 中，上机验证过程如下：

```
-R DS
DS 136E
:2000
-R BX
BX 0000
:1000
-E DS:1000 A0 50 60
-D DS:1000 L3
2000:1000  A0 50 60                                          .P`
-A
```

```
136E:0100 MOV AX,[BX]
136E:0102
-T
AX=50A0  BX=1000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=2000  ES=136E  SS=136E  CS=136E  IP=0102   NV UP EI PL NZ NA PO NC
136E:0102 0000        ADD      [BX+SI],AL                      DS:1000=A0
-
```

3.2.5 寄存器相对寻址方式

寄存器相对寻址方式采用寄存器与位移量的组合，有效地址是寄存器内容与位移量之和。寄存器名只能为 BX、BP、SI、DI 之一，位移量为立即数或符号常量、符号变量，但 DEBUG 中不能使用后两种形式。即：

```
有效地址 = (BX/BP) 或 (SI/DI) + 位移 disp
```

寄存器相对寻址方式的操作数的书写形式，一般有以下几种：

```
[BX]+disp
disp[BX]
[BX+disp]
```

这种寻址方式适于数组、字符串、表格的处理，其地址形成概括为表 3-3。

表 3-3 寄存器相对寻址的地址形成

变　址　器	书 写 形 式	EA	默认段寄存器	物 理 地 址
SI	disp[SI]	(SI)+disp	DS	(DS) * 10H+(SI)+disp
DI	disp[DI]	(DI)+disp	DS	(DS) * 10H+(DI)+disp
BX	disp[BX]	(BX)+disp	DS	(DS) * 10H+(BX)+disp
BP	disp[BP]	(BP)+disp	SS	(SS) * 10H+(BP)+disp

【例 3-6】 说明“MOV　AX,COUNT[SI]”或“MOV　AX,[COUNT+SI]”指令的执行情况。

假设(DS)=3000H,(SI)=2000H,COUNT=10H,则 PA = 32010H；若(32010H)=3EH,(32011H)= 1AH　那么(AX)=1A3EH。相对寻址示意图如图 3-5 所示。

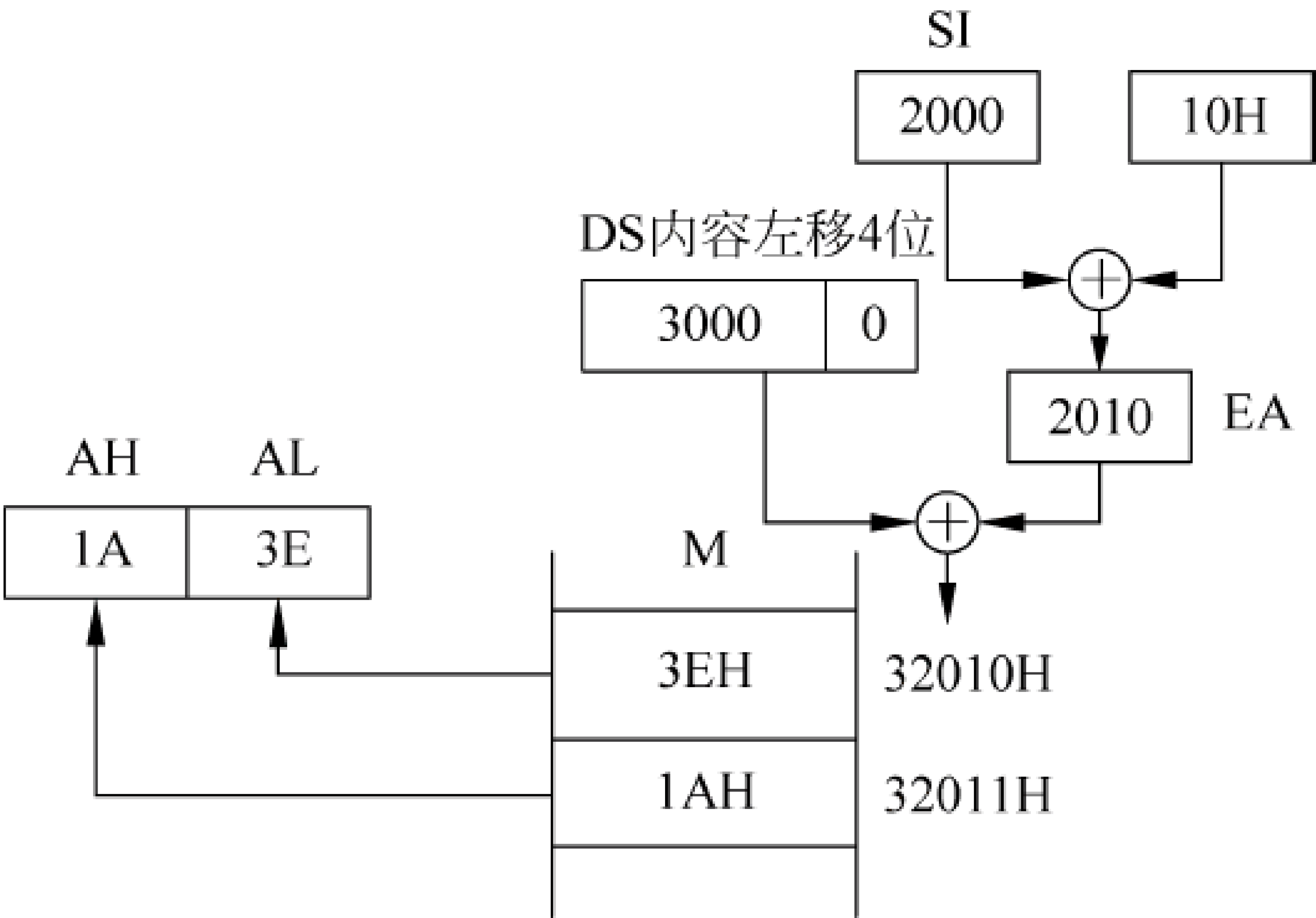


图 3-5　寄存器相对寻址示意图

在 DEBUG 中，进行该指令的验证，上机操作过程如下：

```
-R DS
DS 136E
:3000
-R SI
SI 0000
:2000
-E DS:2010 3E 1A
-D DS:2010 2011
3000:2010  3E 1A                                                    >.
-A
136E:0100 MOV AX,10[SI]
136E:0103
-T
AX=1A3E  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=2000  DI=0000
DS=3000  ES=136E  SS=136E  CS=136E  IP=0103   NV UP EI PL NZ NA PO NC
136E:0103 0000        ADD      [BX+SI],AL                       DS:2000=00
-
```

3.2.6　基址变址寻址方式

基址变址寻址方式是利用基址寄存器(BX/BP)与变址寄存器(SI/DI)的组合，有效地址是基址寄存器内容与变址寄存器内容之和。一般形式为[BX] [SI]、[BX] [DI]、[BP] [SI]或者[BP] [DI]。两个寄存器也可以放在一个[ ]内，用加号连接，如[BX + SI]等。

注意，这种寻址方式只能是一个基址寄存器和一个变址寄存器的组合，即

```
有效地址 = 基址寄存器(BX/BP) + 变址寄存器(SI/DI)
```

【例 3-7】 说明“MOV　DX,[BP+DI]”或“MOV　DX,[BP][DI]”指令的执行情况。

若 SS＝3000H，DS＝2000H，BP＝2000H，DI＝0500H，则有效地址为

```
EA = BP + DI = 2000H + 0500H = 2500H;
```

由于基址寄存器采用 BP，因此默认段寄存器是 SS，这样源操作数的物理地址为

SS×16＋EA＝30000H＋2500H＝32500H。

若指令执行前：DX＝1234H，(32500H)＝78H，(32501H)＝56H；

则指令执行后：DX＝5678H，(32500H)＝78H，(32501H)＝56H，SS＝3000H，DS＝2000H，BP＝2000H，DI＝0500H。

3.2.7　相对基址变址寻址方式

相对基址变址寻址方式又称基址变址相对寻址，它使用基址寄存器、变址寄存器和相对位移量三个组成部分，有效地址是基址寄存器(BX/BP)、变址寄存器(SI/DI)与位移量之和。一般形式是基址变址寻址形式中再增加一个位移量，如[BX][SI] ＋位移量的形式。

```
有效地址 = 基址寄存器(BX/BP) + 变址寄存器(SI/DI) + 位移 disp
```

书写形式有以下几种：

```
[BX][SI] + disp
disp[BX][SI]
[BX + SI] + disp
[BX + SI + disp]
```

当位移量为 0 时，就成为了基址变址寻址方式。其地址的形成过程概括为表 3-4。

表 3-4 相对基址变址寻址的地址形成

基址	变址	书写形式	EA	默认段寄存器	物理地址
BX	SI	disp[BX][SI]	(BX)+(SI)+disp	DS	(DS) * 10H+EA
BX	DI	disp[BX][DI]	(BX)+(DI)+disp	DS	(DS) * 10H+EA
BP	SI	disp[BP][SI]	(BP)+(SI)+disp	SS	(SS) * 10H+EA
BP	DI	disp[BP][DI]	(BP)+(DI)+disp	SS	(SS) * 10H+EA

【例 3-8】 说明“MOV AX,DISP[BX][SI]”指令的执行情况。

假如 DS＝4000H，BX＝3000H，SI＝2000H，DISP＝0600H，则源操作数的有效地址 EA＝BX＋SI＋DISP＝3000H＋2000H＋0600H＝5600H；

其物理地址＝DS×16＋EA＝40000H＋5600H＝45600H。

若执行前：AX＝673AH，(45600H)＝83H，(45601H)＝6AH；

则执行后：AX＝6A83H，(45600H)＝83H，(45601H)＝6AH。

3.2.8 寻址方式小结

表 3-5 概括了上述几种寻址方式，供复习使用。在 DEBUG 下，学习和理解寻址方式有助于深入理解汇编指令及其寻址方式的内涵，真正做到理论与实践相结合。

表 3-5 寻址方式小结

寻 址 方 式	操作数地址	指令格式举例
立即寻址	操作数由指令给出	MOV DX，100H
寄存器寻址	操作数在寄存器中	MOV AX，BX
直接寻址	操作数有效地址由指令直接给出	MOV AX，[2000H]
寄存器间接寻址	见表 3-2	MOV AX，[BX]
寄存器相对寻址	见表 3-3	MOV AL，BUF[SI]
基址变址寻址	见表 3-4	MOV AX,[BX+SI]
相对基址变址寻址	见表 3-4	MOV AX,DISP[BX+SI]

3.3 实 验 内 容

【实验目的】 通过上机操作，理解 8086 CPU 常用的几种寻址方式，特别是存储单元逻辑地址的表示及指令的执行结果，熟练掌握 D、E、R、A、T 等 DEBUG 命令的用法。

【实验 3-1】 现有 DS＝2000H，BX＝0100H，SI＝0002H，(20100H)＝12H，(20101H)＝34H，(20102H)＝56H，(20103H)＝78H，(21200H)＝2AH，(21201H)＝4CH，(21202H)＝0B7H，(21203H)＝65H，试说明下列各条指令的寻址方式及执行后 AX 寄存器的内容。要

求通过 DEBUG 上机验证，比较不同寻址方式的特点，说明上机操作方法和运行结果。

(1) MOV AX,2000H　　;________寻址方式;AX = ________

(2) MOV AX,BX　　;________寻址方式;AX = ________

(3) MOV AX,[1200H]　　;________寻址方式;AX = ________

(4) MOV AX,[BX]　　;________寻址方式;AX = ________

(5) MOV AX,1100H[BX]　　;________寻址方式;AX = ________

(6) MOV AX,[BX + SI]　　;________寻址方式;AX = ________

(7) MOV AX,1100H[BX][SI]　　;________寻址方式;AX = ________

【实验 3-2】 已知(60000H)=12H，(60001H)=34H，(60002H)=56H，(60003H)=78H，(70000H)=0ABH，(70001H)=0CDH，(70002H)=0EFH，(70003H)=0DH。分别执行下列指令后，填入指定寄存器的当前内容，并通过 DEBUG 上机验证，说明上机操作方法和运行结果。

```
MOV   AX, 6000H
MOV   DS, AX
MOV   AX, 7000H
MOV   SS, AX
MOV   BX, 0
MOV   BP, 0
MOV   SI, 2
MOV   AX, BX                ;AX = ________
MOV   AX, [0000H]           ;AX = ________
MOV   AL, [0000H]           ;AL = ________
MOV   AX, [BX]              ;AX = ________
MOV   AL, [BX + 1]          ;AL = ________
MOV   AX, [BX + 1]          ;AX = ________
MOV   AL, [BX + 2]          ;AL = ________
MOV   AX, [BX + 2]          ;AX = ________
MOV   AX, [BX + SI]         ;AX = ________
MOV   AX, [BP + SI]         ;AX = ________
```

习　　题

1. 写出下列指令中存储器操作数的物理地址表达式。

(1) MOV　AL,[BX + 5]

(2) MOV　3[BP],AX

(3) MOV　AX,BUFFER[BX + DI]

(4) MOV　DL,ES:[SI]

2. 8086 系统有哪几种基本的寻址方式？立即寻址和直接寻址有什么不同？寄存器寻址和寄存器间接寻址有什么不同？

3. 已知 DS=2000H，SS=3000H，BX=1234H，BP=0045H，SI=1030H，DISP=25H，说明下列指令中采用的寻址方式，并计算出存储器操作数的物理地址。

（1）MOV AX,[BX]

（2）MOV AX,[BX + DISP]

（3）MOV AX,[BP][SI]

（4）MOV AX,DISP[BX][SI]

4．下列指令执行后，寄存器 AX 和 BX 内容各是什么？要求通过 DEBUG 上机验证。

```
MOV  AX,1234H
MOV  BX,AX
MOV  AH,AL
MOV  [BX],AX
```

第 4 章

程序格式与数据组织

汇编语言程序是用汇编指令及相关的伪操作指令编写的程序，程序书写格式是通过伪指令定义的，数据的组织也是由伪指令来完成。因此，本章主要介绍汇编语言程序格式和数据组织以及相关的伪指令，同时介绍汇编语言程序的上机操作过程。

4.1 程序书写格式

汇编语言程序文件扩展名为.ASM，编程时必须按照一定的格式书写，程序格式是汇编语言程序设计的框架。它有两种格式，一种是完整段定义，另一种是简化段定义。

4.1.1 完整段定义

汇编语言程序是分段的，由若干个段组成，一般是三个段：堆栈段、数据段和代码段。一个程序至少有一个逻辑代码段和 END 伪指令。一个程序文件可以含有多个逻辑数据段、多个逻辑代码段、多个逻辑堆栈段。一般格式书写结构如下：

```
<堆栈段名>  SEGMENT   [STACK]
              …
<堆栈段名>  ENDS
<数据段名>  SEGMENT
              …
<数据段名>  ENDS
<代码段名>  SEGMENT
        ASSUME 定义
过程名或起始标号：

              …
<代码段名>  ENDS
END  过程名或起始标号
```

在汇编语言程序的书写结构中，堆栈段、数据段和代码段是逻辑段。程序在汇编、连接后生成的段是物理段，逻辑段和物理段的关系取决于伪指令 ASSUME 的定义。

【例 4-1】 在屏幕上输出一个字符串“Hello，World!”。

```
;完整段定义举例
DATA SEGMENT
  HMessage DB 'Hello, World!',13,10,'$'
DATA ENDS
CODE SEGMENT
```

```
   ASSUME CS:CODE,DS:DATA
START:MOV AX,DATA
       MOV DS,AX
       MOV AH,09H
       MOV DX,OFFSET HMessage
       INT 21H
       MOV AH,4CH
       INT 21H
CODE ENDS
   END START
```

4.1.2 简化段定义

如果使用简化段定义格式，则例 4-1 的程序可以改写成如下程序：

```
;简化段定义 1
.MODEL SMALL
.DATA
    HMessage DB 'Hello, World!',13,10,'$'
.CODE
START:  MOV AX,@DATA
        MOV DS,AX
        MOV AH,9
        MOV DX,OFFSET HMessage
        INT 21H
        MOV AH,4CH
        INT 21H
    END START
```

Masm 6.0 新增加了入口点和出口点伪操作，这样，简化段定义书写格式又可以进一步简化。

```
;简化段定义 2
.MODEL SMALL
.DATA
HMessage DB 'Hello, World!',13,10,'$'
.CODE
START:  .STARTUP
        MOV AH,9
        MOV DX,OFFSET HMessage
        INT 21H
        .EXIT 0
    END START
```

4.1.3 完整段定义中的伪指令

伪指令从书写形式上看与机器指令很相似，但二者截然不同。机器指令是在程序运行期间执行的，每条指令对应着一种特定的操作，汇编后产生对应的机器代码。伪指令不是程序运行期间由计算机执行的，而是汇编程序对程序进行汇编时处理的操作，完成处理器选择、存储模式定义、数据定义、存储器分配、指示程序开始和结束等功能，伪指令汇编后不产

生与之对应的机器代码。根据伪指令的功能，大致分为以下几类：

- 处理器选择伪操作；
- 段定义伪操作；
- 程序开始和结束伪操作；
- 数据定义及存储器分配伪操作；
- 表达式赋值伪操作；
- 地址计数器与对准伪操作；
- 基数控制伪操作。

1. SEGMENT / ENDS

格式：

```
<段名>  SEGMENT  [STACK]
             ⋮
<段名>  ENDS
```

功能：相当于一个逻辑段的前后括号，必须成对出现。<段名>由用户给出，前后必须一致。可选项[STACK]是专为定义逻辑堆栈段时使用。

2. END

格式：END　[过程名|标号]

功能：程序文件结束。

3. ASSUME

格式：ASSUME　段寄存器名：逻辑段名

功能：用于指定某逻辑段应通过哪个段寄存器寻址，但并不把具体值装入相应的段寄存器。实际上，它就是告诉汇编程序已定义的逻辑段与段寄存器之间的对应关系，并不是为段寄存器赋值。

4.1.4　简化段定义中的伪指令

1. 存储模式选择.MODEL

格式：.MODEL　存储模式　[，语言类型]　[，操作系统类型]　[，堆栈选项]

功能：说明简化段所使用的存储模式，指示数据与代码允许使用的长度。其中，存储模式包括 Tiny、Small、Medium、Compact、Large、Huge、Flat 等。小模式 Small 是常用的一种存储模式，所有数据都放入一个物理段（不超过 64KB）中。

存储模式的具体含义如下：

（1）Tiny：数据与指令代码共存于不超过 64KB 的同一个段内，代码与数据皆为近访问。允许程序从 0100H 开始以扩展名为.COM 的文件形式存储，它适合于小程序。

（2）Small：称为小模式，所有数据放入一个物理段（不超过 64KB）中，所有代码放入另一个物理段中，代码段与数据段均为近程。当不与高级语言连接时，一般用 Small 模式就可以了，是一种常用的内存模式。

（3）Medium：代码由多个段组成，一个模块一个段，数据可组合成不超过 64KB 的段组，这样代码可以进行远程访问而数据是近程访问。

（4）Compact：数据可放在多个段中，形成数据可以远程访问；所有代码都存放在不超

过 64KB 的代码段中，形成近程访问。

(5) Large：代码与数据都可以由多个段组成，都是远程访问。

(6) Huge：与 Large 不同之处是允许数据段超过 64KB。

(7) Flat：在 OS/2 或其他保护模式的操作系统下使用，允许使用 32 位偏移量，只在 Masm 6.0 的汇编程序才支持这种存储模式。

语言类型选项，是高级语言与汇编语言的接口，允许高级语言调用汇编语言编写的过程，可用 C、BASIC 等加以说明。

操作系统类型，是指程序运行在哪种操作系统下，可以用 OS-DOS 或 OS-OS2 说明，OS-DOS 是默认项。

堆栈选项，可用 NearStack 或 FarStack 说明。

2. 堆栈段定义.STACK

格式：.STACK [长度]

功能：定义一个堆栈段，并自动对 SS 和 SP 赋初值，默认段名为@STACK。

3. 数据段定义.DATA

格式：.DATA 存储模式 [,语言类型] [,操作系统类型] [,堆栈选项]

功能：定义一个数据段，默认段名为@DATA。

4. 代码段定义.CODE

格式：.CODE [名字]

功能：定义一个代码段。默认段名为@CODE。如果有多个代码段，要加名字来标识。

在简化段中，定义当前段的开始，就意味着前一个段的结束，它隐含使用了 ASSUME 伪指令。

5. 程序入口.STARTUP

格式：.STARTUP

功能：Masm 6.0 新增加的入口点伪操作。

6. 程序出口.EXIT

格式：.EXIT [返回值]

功能：Masm 6.0 新增加的出口点伪操作。

4.1.5 段寄存器的赋值

在完整段定义中，上面已经提到 ASSUME 伪指令并不负责段寄存器的赋值，需要通过一定的指令来完成。这里只有 CS 无须用户干预，CS 寄存器的赋初值是由系统完成的。而对于 DS、ES、SS 三个段寄存器的赋值，要由用户程序完成。

以 DS 为例，段寄存器的赋值一般方法为：

```
MOV   AX,<逻辑段名>
MOV   DS,AX
```

对于简化段定义，数据段隐含的段名是@DATA，因此给 DS 赋值的方法为：

```
MOV   AX,@DATA
MOV   DS,AX
```

4.1.6　汇编语言程序的结束方式

在 DEBUG 中使用 INT 20H 就可以结束程序，但是这种方式仅用在 DEBUG 中，不能用于 DOS 下的汇编语言程序。汇编语言程序的正常结束，通常使用如下两种方式：

1. 采用 4CH DOS 功能调用

```
MOV  AH,4CH
INT  21H
```

2. RET 方式

```
<程序名>  PROC  FAR
           PUSH  DS
           SUB   AX,AX
           PUSH  AX
           …
           RET
<程序名>  ENDP
           …
           END
```

本书中例题大都采用第一种方式，如例 4-1 的示例程序；第二种方式是子程序返回方式，可以根据个人习惯自行选择，效果相同。

4.2　程序中数据的组织

程序中所涉及的数据除立即数、由指令产生的数、通过键盘等读入的数据外，其他数据、中间结果等都需要在程序设计中进行定义和分配。在程序中，需要将这些数据按照一定的形式存储，并给出访问的原则；而对于保存结果用的存储单元，只需要给出访问形式并预留一定的存储空间。

4.2.1　变量的定义和预置

格式：<变量名>　伪操作符　<操作数>

功能：为变量分配存储单元，将初值放入相应的存储单元中。

说明：<变量名>由字母、数字、下画线等字符组成，第一个字符不能是数字。前 31 个字符有效。保留字不能作为变量名。

<操作数>可以是常数、表达式、字符串、?、DUP 等。

伪操作符：DB，DW，DD，DF，DQ，DT。

变量定义举例：

```
A  DB  100            ;A 为一个字节,初值 100
B  DB  100,2 * 3      ;B 值为 100,B + 1 值为 6
C  DB  'ABCD'         ;C 值 41H,C + 1 值 42H,C + 2 值 43H,C + 3 值 44H
D  DB  ?              ;D 为一个字节,值不定
E  DB  23 DUP (0)     ;23 个 0,从 E 开始每个占一个字节
F  DW  2 * 3          ;一个字,即 06H、00H
```

【例 4-2】 数据定义如下，请说明内存分配情况。

```
COUNTER     DB  6
            DB  'A', 'D',0DH, '$ '
TABLE1      DB 21,45H,255,10110111B
```

图 4-1 是例 4-2 的伪指令在汇编后，数据段内存分配情况示意（图中数值均为十六进制）。COUNTER 指示了存储单元的起始地址，字符存储时是存储其相应的 ASCII 码，其他进制数都转化为十六进制存储。

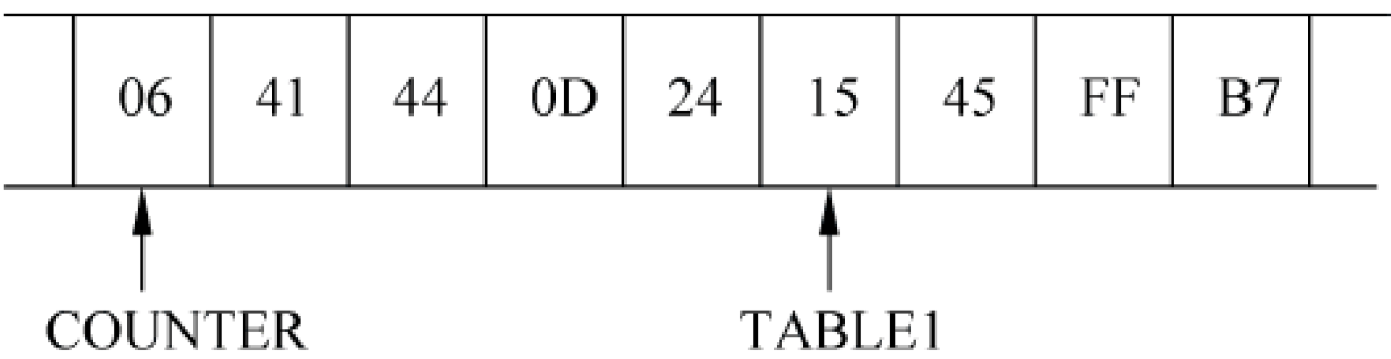


图 4-1 例 4-2 的内存分配情况示意

【例 4-3】 现有如下数据定义，请说明内存分配情况。

```
WORD_VAR     DW  89H,1909H, - 1
             DW  0ABCDH
```

图 4-2 是例 4-3 的伪指令在汇编后，数据段内存分配情况示意（图中数值均为十六进制）。WORD_VAR 指示了存储单元的起始地址，对于 DW 定义的字数据，存储时低位字节存储在低地址，所以存储时是低位在前、高位在后。

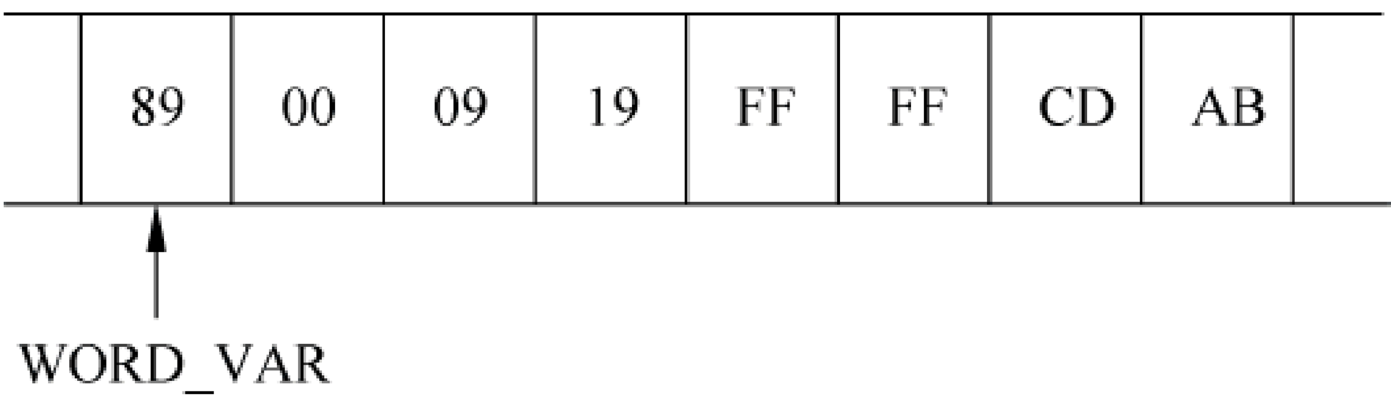


图 4-2 例 4-3 的内存分配情况示意

【例 4-4】 现有如下数据定义，请说明内存分配情况。

```
STR1   DB   'AB'
STR2   DW   'AB'
STR3   DD   'AB'
```

定义字符串必须用单引号引起来，当字符串含有两个以上字符时，要用 DB 定义，这样可以保持原有次序不变，例如 STR1。而用 DW 定义时，两个字符'AB'的 ASCII 码值 4142H 组成一个字数据，存储时就变成了'BA'。用 DD 定义字符串就更不合适了，本例中内存分配情况示意如图 4-3 所示。

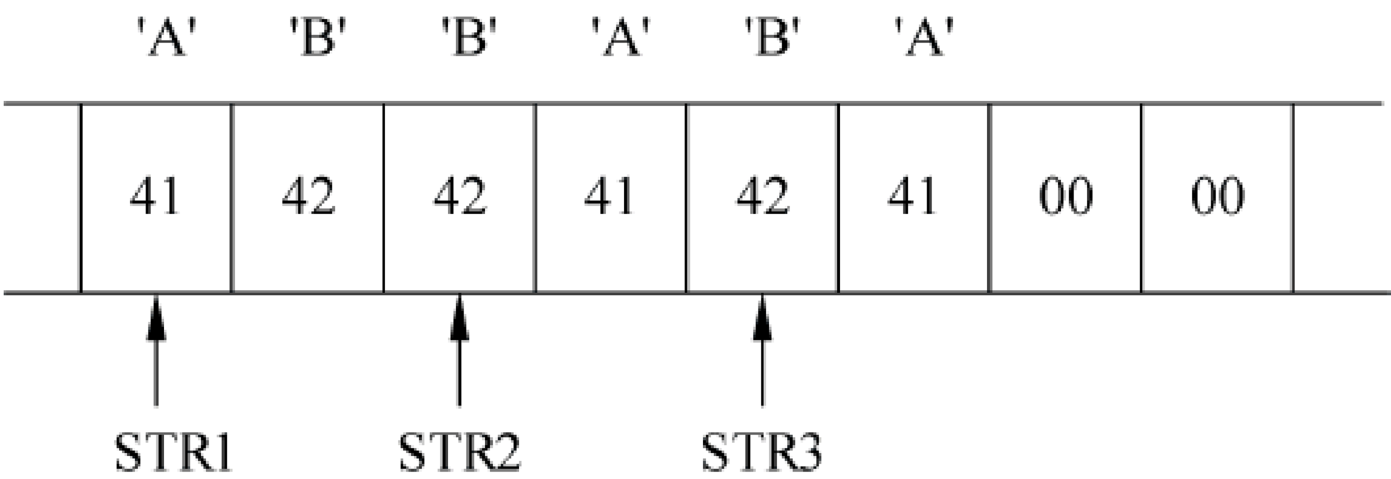


图 4-3 例 4-4 的内存分配情况示意

4.2.2　变量的访问

对变量的访问有多种形式，一般是通过它的名字取其值，还可以通过各种操作符得到其属性值，如段值、位移、类型、长度等。

1. 变量的属性

(1) 段(SEGMENT)：它属于哪个段，段基址是什么。

(2) 位移(OFFSET)：相对于段基址的位移是多少。

(3) 类型(TYPE)：表明数据项的长度，如字节、字、双字、四字等。

2. 数据回送操作符

格式：

```
TYPE   <变量名>               ;DB,DW,DD 分别为 1,2,4
LENGTH  <变量名>              ;取长度,对使用 DUP 的情况,返回变量的单元数; 而对于其他情况,返
                              ;回 1
SIZE   <变量名>               ;返回变量的字节数
SIZE = LENGTH * TYPE
OFFSET  <变量名或标号>        ;取位移值
SEG  <变量名或标号>           ;取段基地址
```

【例 4-5】　逐条运行下列程序，检查每一步有关寄存器的内容，并给出解释。

```
DATA   SEGMENT
    A1   DB   '1234'
    B1   DW   A1
    C1   DD   A1
    D1   DW   0123H, 0F5H
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START: MOV   AX, SEG A1     ;DATA→AX
       MOV   DS, AX
       MOV   AL, LENGTH A1 ;AL←1
       MOV   AH, SIZE A1   ;AH←1
       MOV   BH, TYPE C1   ;BH←4
       MOV   BP, OFFSET C1 ;BP←6
       MOV   AH,4CH
       INT   21H
CODE ENDS
    END START
```

在集成实验环境中，先运行该程序，然后再运行“DEBUG 调试”逐条语句单步执行，可以看到取值操作符已经转换成相应的数值了。运行结果如下：

```
 - T
AX = 13C5   BX = 0000   CX = 0022   DX = 0000   SP = 0000   BP = 0000   SI = 0000   DI = 0000
DS = 13B5   ES = 13B5   SS = 13C5   CS = 13C6   IP = 0003    NV UP EI PL NZ NA PO NC
13C6:0003 8ED8                 MOV       DS,AX
 - T
AX = 13C5   BX = 0000   CX = 0022   DX = 0000   SP = 0000   BP = 0000   SI = 0000   DI = 0000
```

```
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C6   IP = 0005    NV UP EI PL NZ NA PO NC
13C6:0005 B001              MOV       AL,01
-T
AX = 1301   BX = 0000   CX = 0022   DX = 0000   SP = 0000   BP = 0000   SI = 0000   DI = 0000
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C6   IP = 0007    NV UP EI PL NZ NA PO NC
13C6:0007 B401              MOV       AH,01
-T
AX = 0101   BX = 0000   CX = 0022   DX = 0000   SP = 0000   BP = 0000   SI = 0000   DI = 0000
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C6   IP = 0009    NV UP EI PL NZ NA PO NC
13C6:0009 B704              MOV       BH,04
-T
AX = 0101   BX = 0400   CX = 0022   DX = 0000   SP = 0000   BP = 0000   SI = 0000   DI = 0000
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C6   IP = 000B    NV UP EI PL NZ NA PO NC
13C6:000B BD0600            MOV       BP,0006
-T
AX = 0101   BX = 0400   CX = 0022   DX = 0000   SP = 0000   BP = 0006   SI = 0000   DI = 0000
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C6   IP = 000E    NV UP EI PL NZ NA PO NC
13C6:000E B44C              MOV       AH,4C
-
```

变量可以通过它的名字访问，要特别注意操作类型与定义时的类型一致，否则会出现语法错。例如：

```
DATA   SEGMENT
  A  DB   52
  B  DW   'AB'
DATA   ENDS
CODE   SEGMENT
  ASSUME   DS:DATA, CS:CODE
START: MOV   AX, DATA
       MOV   DS, AX
       MOV   AH, A
       MOV   BX, B
       MOV   CX, A     ; 出错
       MOV   CL, B     ; 出错
       …
```

为了提高访问变量的灵活性，可以使用属性操作符，修改变量类型。通过使用属性操作符可以实现。

3. 属性操作符

1）PTR

格式：<类型> PTR <地址表达式>

其中，<类型>可以是 BYTE、WORD、DWORD、FWORD 等。

功能：使地址表达式具有另一种类型属性

例如：

```
MOV  [BX],5
```

必须用 PTR 说明，写成：

```
MOV   BYTE PTR [BX],5
```

或

```
MOV WORD PTR [BX],5
```

2）段操作符

格式 1：<段寄存器名>:<地址表达式>

格式 2：<逻辑段名>:<地址表达式>

功能：指定该有效地址相对指定段寻址。

例如：

```
MOV   AX,ES:[BX + SI]
MOV   BX,SS:[SI + 3]
MOV   BX,DATA2:WORD PTR [100]
```

3）THIS（双重定义）

格式：<变量名 1>　EQU　THIS　<类型 1>
　　　<变量名>　DB | DW | DD…

<类型 1>应和<变量名>使用的定义类型不同。

例如：

```
FIRST_TYPE   EQU   THIS   BYTE
WORD_TABLE   DW   100   DUP(?)
```

FIRST_TYPE 的偏移地址与 WORD_TABLE 完全相同。访问 FIRST_TYPE 时，按字节类型；而访问 WORD_TABLE 时，按字类型。

4. 表达式赋值伪操作

格式：表达式名　EQU　表达式

功能：将表达式的值赋予一个名字，以后在程序中就可以用这个名字来代替表达式。

例如：

```
ALPHA    EQU   9
BETA     EQU   ALPHA + 18
BB       EQU   [BP + 8]
```

与 EQU 类似的"＝"伪操作（允许重复定义）如下：

```
…
EMP = 7
…
EMP = EMP + 1
…
```

5. 定位伪操作

格式：ORG 常数表达式

功能：设置当前地址计数器的值。

【例 4-6】 数据段定义如下：

```
DATA   SEGMENT
  ORG 100H
DA1   DB 20 DUP('A')
  ORG 200H
DA2   DW 20 DUP('B')
DATA   ENDS
```

请说明 DA1 和 DA2 的起始地址是什么？

在这个数据段定义中，DA1 的起始地址不是 0 而是 100H，DA2 的起始地址为 200H，这就是 ORG 指令作用的结果。

6. 地址计数器

在汇编程序对程序汇编的过程中，使用地址计数器来保存当前正在汇编的指令或数据地址。地址计数器值可用＄来表示，汇编语言允许编程者直接用＄来引用地址计数器的当前值。在指令和伪指令中，同样可以直接用＄来表示地址计数器的当前值。

【例 4-7】 输入定义如下：

```
    ORG 0100H
ARRAY   DW   1, 2 , $ +3 , 3 , 4 , $ +4
```

请说明 ARRAY 变量的内存分配情况。

注意：＄值是不断变化的。'＄＋3'中，＄的值为 0104H；而在'＄＋4'中，＄的值为 010AH。其内存分配情况示意如图 4-4 所示。

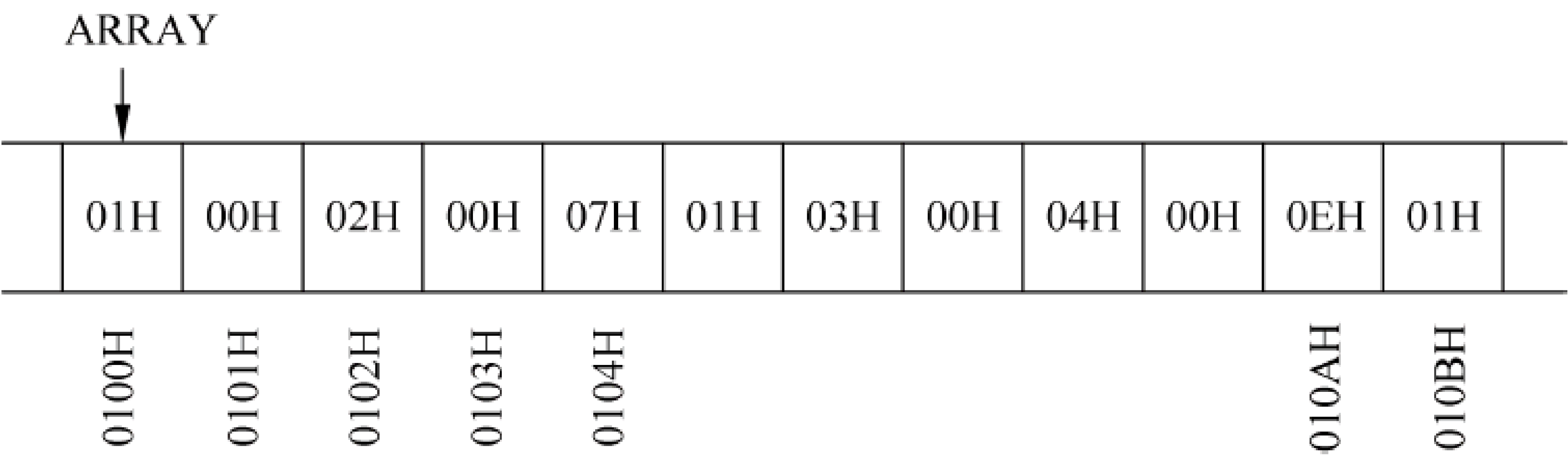


图 4-4 例 4-7 的内存分配情况示意

4.3 汇编语言程序的上机过程

汇编语言程序设计实践性很强，编辑、调试、运行等上机操作是必不可少的手段，其一般上机操作过程，如图 4-5 所示。

1. 编辑

在文本编辑软件中，可以选用 EDIT、Windows 记事本等，建立汇编语言程序，文件的扩展名为.ASM 的纯文本文件。编辑过程中可先进行人工检查和修改，直至观察没有错误时为止，存盘后可以进行下一步——汇编。

2. 汇编

利用汇编程序，将汇编语言程序翻译成目标代码的过程称为汇编。常用的汇编程序有 Masm、Tasm。在对程序文件汇编时，汇编程序将对.ASM 文件进行两遍扫描，若程序文件

中有语法错误，则结束汇编，汇编程序将指出程序中存在的错误，这时应返回文本编辑环境修改程序中的错误。修改后，再重新汇编，直到最后得到无错误的目标程序，即生成扩展名为.OBJ 的目标文件。

3. 连接

汇编后产生的目标程序(.OBJ 文件)，必须通过连接程序(LINK 或 TLINK)连接成一个可执行程序后才能运行。连接程序进行连接时，其输入有两部分：一部分是目标文件(.OBJ)，目标文件可以是一个也可以是多个，可以是汇编语言经汇编后产生的目标文件，也可以是高级语言(例如 C 语言)经编译后产生的目标文件；另一部分是库文件(.LIB)，库文件是系统中已经建立的，主要是为高级语言提供的。连接后输出两个文件，一是扩展名为.EXE 的可执行文件，另一个是扩展名为.MAP 的内存分配文件，它是连接程序的列表文件，又称连接映像(Link Map)，它给出每个段在存储器中的分配情况，该文件可有可无。

4. 运行

通过编辑、汇编和连接后的程序是可以执行的程序，可以在 DOS 下运行，在运行时发现逻辑错误如运行结果不正确，还需要进行动态调试，找到错误后，修改程序，然后再重新进行汇编、连接、运行操作，直至没有错误为止。

5. 调试

对于简单程序，人工分析就可以解决，但是复杂的程序必须借助调试工具进行动态调试，找出问题的根源，回到第一步修改程序，再重新进行汇编、连接等操作。具体调试方法有单步执行、设置断点等动态调试方法，这些需要一定的上机实践经验。常用的调试工具有 DEBUG、Turbo Debugger、Code View Debugger 等。

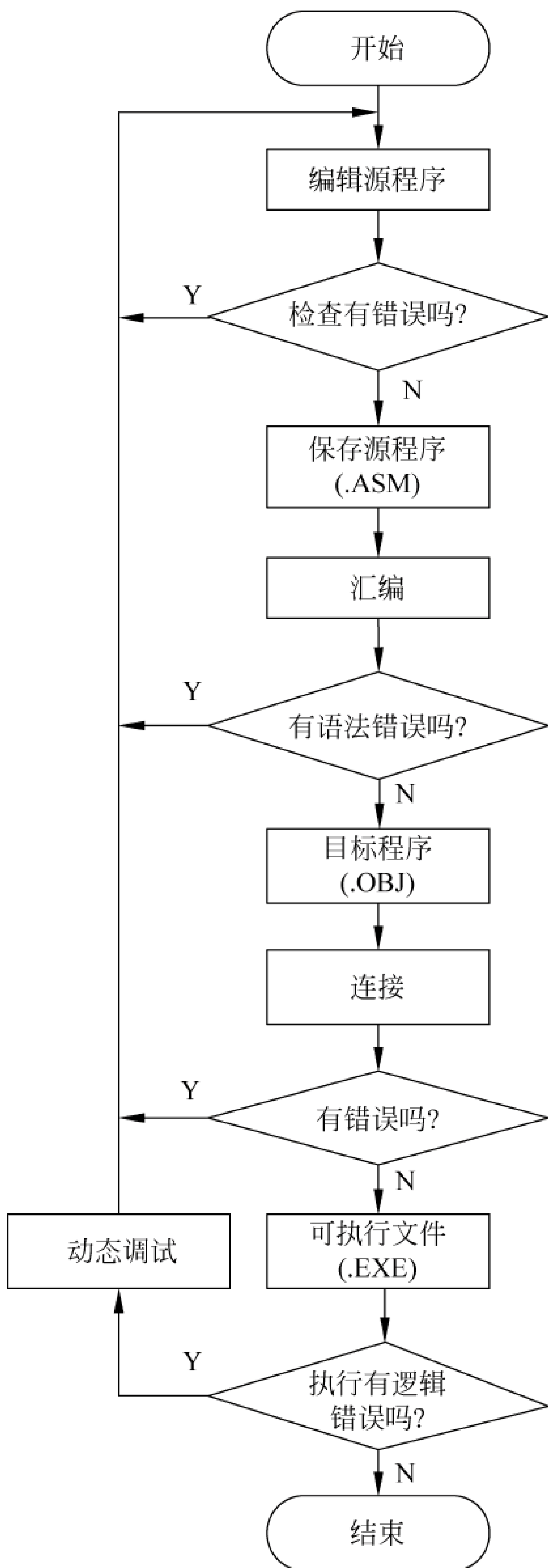


图 4-5　上机操作过程

本书采用 Masm for Windows 集成实验环境，操作方便，简单易学，省去了分步操作的过程，将所有上机操作在一个环境下完成，详细操作方法参见附录 B。

4.4　实验内容

【实验目的】 熟练掌握 Masm for Windows 集成实验环境的操作方法，上机练习验证伪指令操作的结果，并通过实例程序掌握汇编语言程序的上机操作过程。

【实验 4-1】 下列语句在存储器中每个变量应分配到多少字节？画出存储器分配示意图(标出偏移地址)，并上机验证分析结果。

```
DA1   DW   5
DA2   DW 4 DUP(?),2,3
DA3   EQU 10
DA4   DD DA3 DUP(?)
DA5   DB 2 DUP(?,DA3 DUP(2,3))
DA6   DB 'How are you?'
```

解析：对于伪指令，不能直接上机运行，一般需要编写成汇编语言程序，经过汇编、连接、运行后，才能在 DEBUG 下查看运行结果。利用实验 4-1 的伪指令，不妨利用集成实验环境中提供的汇编语言格式模板编写如下程序：

```
;实验 4-1 程序
DATAS SEGMENT
    DA1   DW   5
    DA2   DW 4 DUP(?),2,3
    DA3   EQU 10
    DA4   DD DA3 DUP(?)
    DA5   DB 2 DUP(?,DA3 DUP(2,3))
    DA6   DB 'How are you?'
DATAS ENDS
CODES SEGMENT
    ASSUME CS:CODES,DS:DATAS
START:  MOV AX,DATAS
        MOV DS,AX
        ;此处输入代码段代码
        MOV AH,4CH
        INT 21H
CODES ENDS
        END START
```

在集成实验环境中，先单击工具栏中的“运行”按钮，运行该程序，虽然没有显示输出结果，但是已经生成了 EXE 文件。然后，选择“运行”菜单下的“DEBUG 调试”，利用 T 命令，单步执行前两条语句，即完成段寄存器 DS 装填后，查看数据段的内存分配情况，运行结果如下：

```
-T
AX=13C5  BX=0000  CX=0079  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13CC  IP=0003   NV UP EI PL NZ NA PO NC
13CC:0003 8ED8              MOV     DS,AX
-T
AX=13C5  BX=0000  CX=0079  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13CC  IP=0005   NV UP EI PL NZ NA PO NC
13CC:0005 B44C              MOV     AH,4C
-D DS:0
13C5:0000  05 00 00 00 00 00 00 00-00 00 02 00 03 00 00 00   ..................
13C5:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ..................
13C5:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ..................
```

```
13C5:0030  00 00 00 00 00 00 00 02-03 02 03 02 03 02 03 02   ................
13C5:0040  03 02 03 02 03 02 03 02-03 02 03 00 02 03 02 03   ................
13C5:0050  02 03 02 03 02 03 02 03-02 03 02 03 02 03 02 03   ................
13C5:0060  48 6F 77 20 61 72 65 20-79 6F 75 3F 00 00 00 00   How are you?....
13C5:0070  B8 C5 13 8E D8 B4 4C CD-21 00 00 00 00 00 00 00   .....L.!........
-
```

【实验 4-2】 上机完成例 4-5 的程序，单步执行，查看每条指令的运行结果。

解析：在集成实验环境中编辑例 4-5 的程序，经过汇编、连接、运行没有错误时，运用 DEBUG 调试单步执行。先运行前两条段寄存器装填指令，然后查看存储器情况，运行结果如下：

```
-T
AX=13C5  BX=0000  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13C6  IP=0003   NV UP EI PL NZ NA PO NC
13C6:0003 8ED8          MOV     DS,AX
-T
AX=13C5  BX=0000  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0005   NV UP EI PL NZ NA PO NC
13C6:0005 B001          MOV     AL,01
-D DS:0
13C5:0000  31 32 33 34 00 00 00 00-C5 13 23 01 F5 00 00 00   1234......#.....
13C5:0010  B8 C5 13 8E D8 B0 01 B4-01 B7 04 BD 06 00 B4 4C   ...............L
13C5:0020  CD 21 00 00 00 00 00 00-00 00 00 00 00 00 00 00   .!..............
13C5:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

然后，再接着单步执行，可以看到数据回送操作符都已经显示为数据了。LENGTH A1 的值就是 01H，所以，下一条指令是“MOV AL,01H”。逐条语句运行，查看运行结果如下：

```
-T
AX=1301  BX=0000  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0007   NV UP EI PL NZ NA PO NC
13C6:0007 B401          MOV     AH,01
-T
AX=0101  BX=0000  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0009   NV UP EI PL NZ NA PO NC
13C6:0009 B704          MOV     BH,04
-T
AX=0101  BX=0400  CX=0022  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=000B   NV UP EI PL NZ NA PO NC
13C6:000B BD0600        MOV     BP,0006
-T
AX=0101  BX=0400  CX=0022  DX=0000  SP=0000  BP=0006  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=000E   NV UP EI PL NZ NA PO NC
13C6:000E B44C          MOV     AH,4C
-
```

在单步执行过程中，遇到 INT 21H 命令，选用 P 命令执行。若用 T 命令，它会跟踪到中断程序内部，没有必要。这也是 P 命令与 T 命令的区别。

```
-T
AX=4C01  BX=0400  CX=0022  DX=0000  SP=0000  BP=0006  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0010   NV UP EI PL NZ NA PO NC
13C6:0010 CD21            INT     21
-P
Program terminated normally
-
```

习　　题

1. 以 DA1 为首地址的数据区中，存放以下数据："A","B",0,0,"C","D",0,0，请分别用 DB、DW 和 DD 实现。

2. 定义一个数据段完成以下操作序列，并上机验证伪指令的正确性，写出实验操作方法。

(1) 将 30H、48、2AH、100 存放在字节变量 DA1 为首地址的存储单元中；

(2) 将字数据 7852H、1254、85H、0B4H 存放在字节变量 DA2 为首地址的存储单元中，而且不可改变字数据高低字节存放的约定；

(3) 将字节数据 56H、23H、89H、3CH 存放在字变量 DA3 为首地址的存储单元中，而且不可改变数据按字节存储的次序；

(4) 在 DA4 为首地址的字节存储单元中连续存放指定的字节数据：9 个'a'，2 个 38，6 个(5,7)，8 个空格单元；

(5) 在 DA5 为首地址的存储单元中存放字符串"Exercise!"；

(6) 用符号 COUNT 代替 100。

3. 已知数据定义如下：

```
    ORG 0100H
BLOCK1  DW 1234H,5678H,9ABCH
BLOCK2  DB 38H
CNT1    DW BLOCK1
CNT2    EQU BLOCK2- BLOCK1
```

请问 CNT1 的内容和 CNT2 代表的值各是什么？

4. 已知数据定义如下：

```
BUF1  DB 1AH, 2BH
BUF2  DW 3CH, 4DH, 5678
BUF3  DB 2, 100 DUP(?)
BUF4  DB 'ABCDEF'
BUF5  DW BUF3
BUF6  EQU $ - BUF3
```

请问：

（1）该数据定义在存储器中应分配多少字节？

（2）BUF5 单元的内容是什么？

（3）BUF6 的值是多少？

（4）执行“MOV AL,BUF4+2”指令后,AL 的内容是什么？

（5）执行“MOV AX，WORD PTR BUF1”指令后,AX 的内容是什么？

（6）上机验证你所得到的结果,说明操作步骤。

5. 在代码段中编写如下程序：

```
MOV   DL,'5'
MOV   AH,2
INT   21H
```

（1）用完整段定义格式书写汇编语言程序,并在集成实验环境下运行。

（2）用简化段定义格式书写汇编语言程序,并在集成实验环境下运行。

（3）运行例 4-1 的程序,查看运行结果。

（4）归纳汇编语言程序的书写格式。

6. 对于给出的数据段定义,写出下列各条指令执行后的结果并上机验证,说明上机操作步骤。

```
DATA SEGMENT
 DA1 DB ?,10 DUP(?)
 DA2 DW 20 DUP(0)
 DA3 DB 'ABCDEF'
DATA ENDS
     …
     MOV AX, TYPE DA1
     MOV BX, TYPE DA2
     MOV CX, LENGTH DA2
     MOV SI, SIZE DA2
     MOV DI, LENGTH DA3
     …
```

第5章

数据传送程序

在前面章节中，初步介绍了 MOV 指令的用法，本章将详细地介绍数据传送指令及其在程序设计中的应用。通过学习，要求熟练掌握 MOV、PUSH、POP、XCHG 等指令的功能，并了解其他数据传送指令的用法，通过上机调试数据传送程序，使学生具有初步的顺序结构程序设计能力。

5.1 数据传送

5.1.1 数据传送指令分类

数据传送指令的主要功能是把数据、地址等信息传送到寄存器或存储单元中，通常分为以下几类：

- 通用数据传送指令：MOV、PUSH、POP、XCHG；
- 累加器专用传送指令：IN、OUT、XLAT；
- 地址传送指令：LEA、LDS、LES；
- 标志寄存器传送指令：LAHF、SAHF、PUSHF、POPF。

5.1.2 MOV 指令

格式：`MOV   DST,SRC`

功能：数据从源操作数(SRC)的副本传送到目的操作数(DST)中，它不影响源操作数，也不影响状态标志。

在 MOV 指令中，大多数操作数均可以作为源和目的操作数，但是，立即数和 CS 不能作为目的操作数；不能在两个存储单元之间直接传送；不允许在两个段寄存器之间直接传送。

MOV 指令应用分下列情形：

(1) 寄存器与寄存器之间的数据传送(CS 和 IP 除外)。例如：

```
MOV   AX,BX
MOV   DL,AH
MOV   BP,SI
```

(2) 立即数传送到通用寄存器(注意，立即数不能直接传送到段寄存器)。例如：

```
MOV   AX,100Q
MOV   AL,4
```

(3) 寄存器(CS 和 IP 除外)与存储器之间。例如:

```
MOV  AL,BUFFER
MOV  [BX][DI],DL
MOV  SI,ES:[BP]
```

(4) 立即数传送到存储器,注意类型的一致性。例如:

```
MOV  BYTE PTR [SI],5
MOV  WORD PTR [BX],6
```

MOV 指令对大多数操作数对的结合都是合法的,但是对以下情况不能用 MOV 直接实现:

(1) 两个存储单元之间数据传送不能直接实现,一般通过间接方式实现。如,将 BUFFER1 的一个字节数据传送到 BUFFER2 单元中:

```
MOV  AL,BUFFER1
MOV  BUFFER2,AL
```

(2) 不能将立即数直接装入段寄存器,通常用以下形式:

```
MOV  AX,1020H
MOV  DS,AX
```

(3) 不能将一个段寄存器内容直接传送到另一个段寄存器,通常用以下形式:

```
MOV  AX,DS
MOV  ES,AX
```

【例 5-1】 存储器结构如图 5-1 所示,DS=2000H,执行下列指令后,存储单元内容有何变化?(AX)=?要求在 DEBUG 下验证结果。

```
MOV  BX,7000H
MOV  SI,8000H
MOV  AL,40H
MOV  [BX],AL
MOV  AL,[SI]
MOV  AH,AL
```

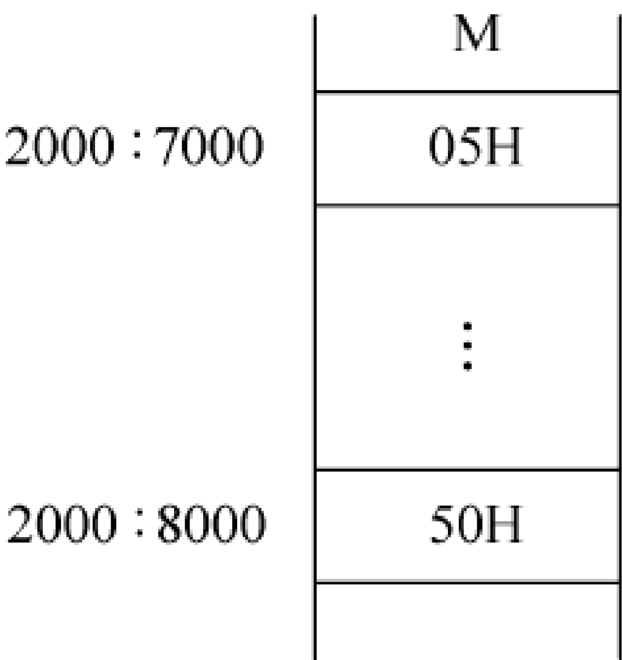


图 5-1　存储结构示意图

在例 5-1 中,前 3 条语句都是立即数传送,执行后,BX=7000H,SI=8000H,AL=40H;第 4 条语句,功能是将 AL 的内容传送到 BX 所指定的存储单元中,即将 40H 放入(2000:7000)单元;第 5 条语句,是将 SI 所指定的存储单元内容送到寄存器 AL 中,即将(2000:8000)单元的 50H 送到 AL,使 AL=50H,存储单元内容不变;最后一条语句,是寄存器之间的数据传送,结果 AH=50H,这样 AX=5050H。

在 DEBUG 下上机验证:首先,用 R 命令设置 DS 寄存器的初始值,用 E 命令设置有关存储单元的内容;然后,用 A 命令将上述汇编指令进行汇编,用 T 命令逐条语句单步执行,同时查看寄存器和存储器内容的变化。具体操作方法如下:

```
-R DS
```

```
DS 136E
:2000
-E DS:7000 05
-E DS:8000 50
-A
136E:0100 MOV BX,7000
136E:0103 MOV SI,8000
136E:0106 MOV [BX],AL
136E:0108 MOV AL,[SI]
136E:010A MOV AH,AL
136E:010C
-T
AX=0000  BX=7000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=2000  ES=136E  SS=136E  CS=136E  IP=0103   NV UP EI PL NZ NA PO NC
136E:0103 BE0080          MOV     SI,8000
-T
AX=0000  BX=7000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=8000  DI=0000
DS=2000  ES=136E  SS=136E  CS=136E  IP=0106   NV UP EI PL NZ NA PO NC
136E:0106 8807            MOV     [BX],AL                     DS:7000=05
-T
AX=0000  BX=7000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=8000  DI=0000
DS=2000  ES=136E  SS=136E  CS=136E  IP=0108   NV UP EI PL NZ NA PO NC
136E:0108 8A04            MOV     AL,[SI]                     DS:8000=50
-T
AX=0050  BX=7000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=8000  DI=0000
DS=2000  ES=136E  SS=136E  CS=136E  IP=010A   NV UP EI PL NZ NA PO NC
136E:010A 88C4            MOV     AH,AL
-T
AX=5050  BX=7000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=8000  DI=0000
DS=2000  ES=136E  SS=136E  CS=136E  IP=010C   NV UP EI PL NZ NA PO NC
136E:010C 0000            ADD     [BX+SI],AL                  DS:F000=00
-
```

【例 5-2】 下列指令执行后,(AX)=?(BX)=?(DS:3040H)=? 要求用集成实验环境编写程序验证。

```
MOV  AX,1020H
MOV  DS,AX
MOV  AX,0ABCH
MOV  BX,3040H
MOV  [BX],AL
MOV  AL,BL
```

在例 5-2 中,前两条指令执行后,DS=1020H;紧接着的两条指令都是立即数给寄存器赋值,结果是 AX=0ABCH,BX=3040H;第 5 条指令是寄存器间接寻址,将 AL 的内容传送到 BX 指定的存储单元中,即(DS:3040H)=BCH;最后是 BL 的内容传送到 AL,使得 AL=40H。最终结果是:AX=0A40H,BX=3040H,(DS:3040H)=BCH。

在集成实验环境下,可以利用上述汇编指令编写程序如下:

```
CODES SEGMENT
```

```
    ASSUME CS:CODES
START:MOV   AX,1020H
      MOV   DS,AX
      MOV   AX,0ABCH
      MOV   BX,3040H
      MOV   [BX],AL
      MOV   AL,BL
      MOV   AH,4CH
      INT   21H
CODES ENDS
    END START
```

经过汇编、连接后，可以在DEBUG下单步执行，查看每条指令执行后寄存器或存储器的内容变化。运行结果如下：

```
-T
AX=1020  BX=0000  CX=0013  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13C5  IP=0003  NV UP EI PL NZ NA PO NC
13C5:0003 8ED8          MOV    DS,AX
-T
AX=1020  BX=0000  CX=0013  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1020  ES=13B5  SS=13C5  CS=13C5  IP=0005  NV UP EI PL NZ NA PO NC
13C5:0005 B8BC0A        MOV    AX,0ABC
-T
AX=0ABC  BX=0000  CX=0013  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1020  ES=13B5  SS=13C5  CS=13C5  IP=0008  NV UP EI PL NZ NA PO NC
13C5:0008 BB4030        MOV    BX,3040
-T
AX=0ABC  BX=3040  CX=0013  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1020  ES=13B5  SS=13C5  CS=13C5  IP=000B  NV UP EI PL NZ NA PO NC
13C5:000B 8807          MOV    [BX],AL                       DS:3040=00
-T
AX=0ABC  BX=3040  CX=0013  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1020  ES=13B5  SS=13C5  CS=13C5  IP=000D  NV UP EI PL NZ NA PO NC
13C5:000D 8AC3          MOV    AL,BL
-T
AX=0A40  BX=3040  CX=0013  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1020  ES=13B5  SS=13C5  CS=13C5  IP=000F  NV UP EI PL NZ NA PO NC
13C5:000F B44C          MOV    AH,4C
-D DS:3040
1020:3040  BC 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1020:3050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1020:3060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1020:3070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1020:3080  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1020:3090  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1020:30A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1020:30B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

上面分别用了两种方法验证汇编指令的执行结果，虽然方法略有不同，但实质都是一样的，在实际上可以自由选用。

5.1.3 堆栈操作

1. 进栈指令

格式：PUSH SRC

功能：将源操作数(SRC)的值压入堆栈。

执行操作：(SP) ← (SP) − 2

((SP)+1, (SP)) ←(SRC)

执行过程：

(1) 先(SP) ←(SP)−1,然后将(SRC)高位字节送至 SP 所指单元;

(2) 再(SP) ← (SP)−1,将(SRC)低位字节送至 SP 所指单元。

【例 5-3】 若(SP)=2008H,(AX)=1020H,执行前情况如图 5-2 所示。指令 PUSH AX 执行后,寄存器和堆栈内容如何变化?

执行 PUSH AX 后,堆栈如图 5-3 所示,SP 指向 2006H 单元,(AX)内容不变。

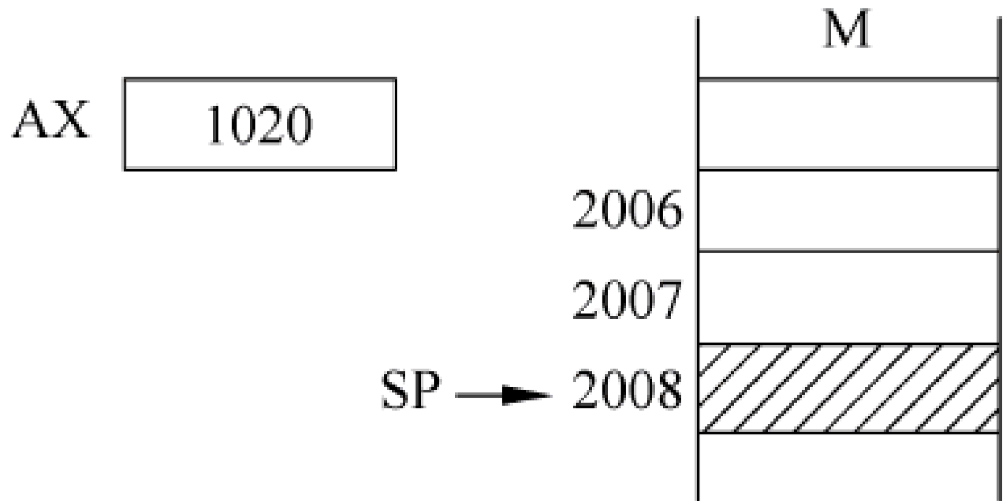


图 5-2 PUSH 指令执行前示意图

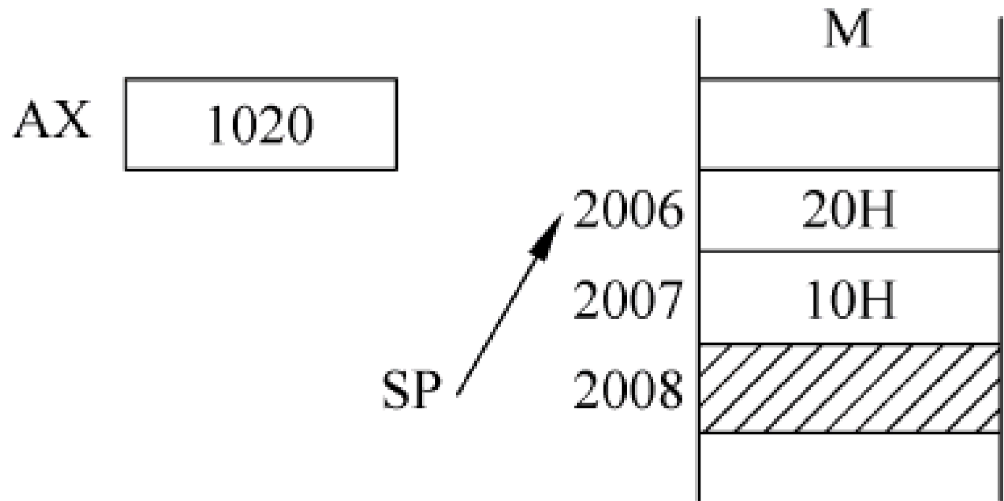


图 5-3 PUSH 指令执行后示意图

2. 出栈指令

格式：POP DST

功能：从栈顶弹出一个字送给目的操作数(DST)。

执行操作：(DST) ← ((SP)+1, (SP))

(SP)← (SP) + 2

执行过程：

(1) 先将 SP 所指单元内容弹出送至 DST 的低位字节,然后(SP) ← (SP) + 1;

(2) 再将 SP 所指单元内容弹出送至 DST 的高位字节,然后(SP) ← (SP) + 1。

例如,假设执行前堆栈的初始状态如图 5-3 所示,执行“POP BX”后,从栈顶单元弹出一个字送入 BX,使得 BX=1020H,SP=2008H,POP 指令执行后如图 5-4 所示。

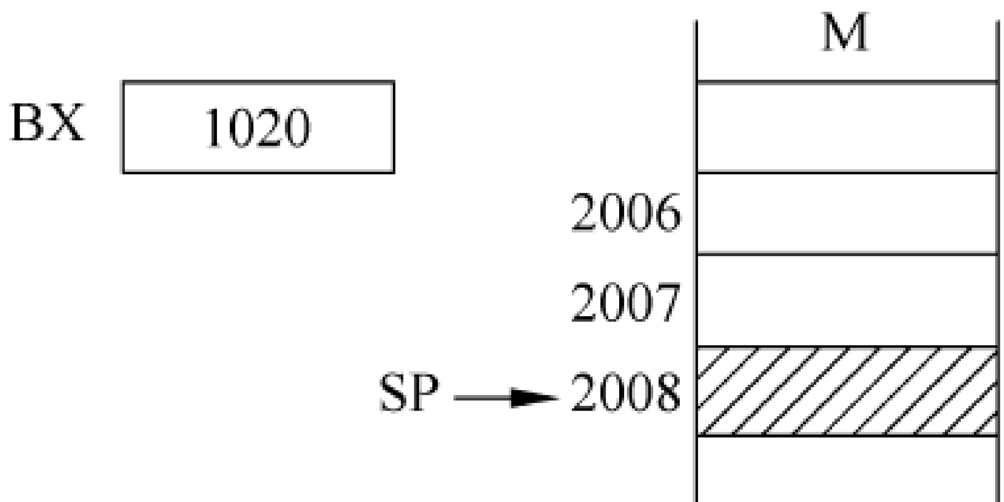


图 5-4 POP 指令执行后示意图

PUSH/POP 指令的操作过程可以在 DEBUG 环境下验证,操作前先设置好 SP 和 AX 初始值,然后单步执行指令,查看寄存器 AX、SP 和堆栈中数据的变化,具体操作过程和运行结果如下:

```
-R AX
AX 0000
:1020
```

```
-R SP
SP FFEE
:2008
-A
136E:0100 PUSH AX
136E:0101 POP BX
136E:0102
-T
AX=1020  BX=0000  CX=0000  DX=0000  SP=2006  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0101   NV UP EI PL NZ NA PO NC
136E:0101 5B              POP      BX
-D SS:2006 2007
136E:2000                      20 10                                      .
-T
AX=1020  BX=1020  CX=0000  DX=0000  SP=2008  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0102   NV UP EI PL NZ NA PO NC
136E:0102 0000            ADD      [BX+SI],AL              DS:1020=00
-D SS:2006 2007
136E:2000                      D2 0D                              ..
-
```

从运行结果可以看出，执行 PUSH AX 后，堆栈指针 SP 减 2，寄存器 AX 的内容压入堆栈，AX 内容不变；当执行 POP BX 后，堆栈弹出一个字送入 BX，SP 加 2，这时栈顶单元是(SS:2008)，若再查看(SS:2006)和(SS:2007)单元，已经没有意义，因为它不是栈空间了。

3. 几点说明

堆栈是"后出先进"的存储区，段地址存放在 SS 中。SP 在任何时候都指向栈顶，进出栈后系统会自动修改 SP 寄存器的值。

(1) 8086 堆栈操作必须以字为单位。

(2) PUSH/POP 不影响标志位。

(3) 不能用立即寻址方式，如"PUSH　1234H"错误。

(4) DST 不能是 CS，如"POP　CS"错误。

利用堆栈，可以进行现场数据保护和子程序返回。例如，要保护寄存器 AX、BX、CX 的内容，可以将它们先压入堆栈，在适当的时候再弹出，起到保护作用。使用时注意入栈和出栈的次序，一般格式如下：

```
PUSH  AX
PUSH  BX
PUSH  CX
 …                          ;其间用到 AX、BX、CX
POP   CX                    ;后进先出
POP   BX
POP   AX
```

5.1.4　交换指令

格式：XCHG　操作数 1，操作数 2

执行操作：(操作数 1)和(操作数 2)内容互换。

说明：

(1) 可实现寄存器之间，寄存器与存储器之间的数据交换，但不能直接实现两个存储器的内容互换。

例如：

```
XCHG   AX,BX
XCHG   AL,BH
XCHG   BX,DATA[SI]
```

(2) 不允许使用段寄存器。

例如“XCHG　DS,ES”错误，应改为：

```
MOV   AX,DS
MOV   BX,ES
MOV   DS,BX
MOV   ES,AX
```

(3) XCHG 指令不影响标志位。

【例 5-4】 写出进行存储器字变量 A 和存储器字变量 B 的内容互换的指令序列；假设 A 的值为 4241H，B 的值为 4443H，编程实现 A 和 B 互换，写出汇编语言程序。

由于两个存储器操作数之间不能直接实现互换操作，因此只能用间接方式实现。一般采用如下两种方法：

方法 1：

```
MOV   AX,   B
XCHG   AX,   A
MOV   B,   AX
```

方法 2：

```
PUSH   A
PUSH   B
POP   A
POP   B
```

编程时，采用完整段定义方法，在数据段定义两个字变量，代码段可以采用上述的任意一种方法。书写的汇编语言程序如下：

```
DATA SEGMENT
   A DW 4241H
   B DW 4443H
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:   MOV AX,DATA
         MOV DS,AX
         MOV AX,B
         XCHG AX,A
         MOV B,AX
         MOV AH,4CH
```

```
        INT 21H
CODE ENDS
    END START
```

在集成实验环境中，采用 DEBUG 调试，单步执行程序，可以查看数据交换前存储器的数据和交换后存储器的情况，对比发现，实现字变量 A 和 B 的互换。操作步骤和运行结果如下：

```
-T
AX=13C5  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13C6  IP=0003   NV UP EI PL NZ NA PO NC
13C6:0003 8ED8          MOV     DS,AX
-T
AX=13C5  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0005   NV UP EI PL NZ NA PO NC
13C6:0005 A10200        MOV     AX,[0002]               DS:0002=4443
-D DS:0 L4
13C5:0000  41 42 43 44                                      ABCD
-T
AX=4443  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0008   NV UP EI PL NZ NA PO NC
13C6:0008 87060000      XCHG    AX,[0000]               DS:0000=4241
-T
AX=4241  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=000C   NV UP EI PL NZ NA PO NC
13C6:000C A30200        MOV     [0002],AX               DS:0002=4443
-T
AX=4241  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=000F   NV UP EI PL NZ NA PO NC
13C6:000F B44C          MOV     AH,4C
-D DS:0 L4
13C5:0000  43 44 41 42                                      CDAB
-
```

5.2 换码指令

格式：XLAT <换码表>

功能：用换码表中的一个字节来置换 AL 中的内容。

执行操作：(AL) ← ((BX)+(AL))

入口参数：换码表起始地址在 BX 中，换码字节在换码表中的位移在 AL 中。

出口参数：置换后的内容在 AL 中。

注意：

- 不影响标志位；
- 字节表格(长度不超过 256)；
- 首地址送给 BX 寄存器；
- 需转换的代码位移量送给 AL 寄存器。

【例 5-5】 换码表 TABLE 的存储情况如图 5-5 所示，说明下列指令序列中 XLAT 指令执行前后寄存器 AL 内容的变化。

```
MOV BX, OFFSET  TABLE
MOV  AL, 4
XLAT  TABLE
```

指令执行前，BX 指向了换码表的首地址 TABLE，AL＝04H；指令执行后，AL＝34H，执行过程示意图如图 5-5 所示。

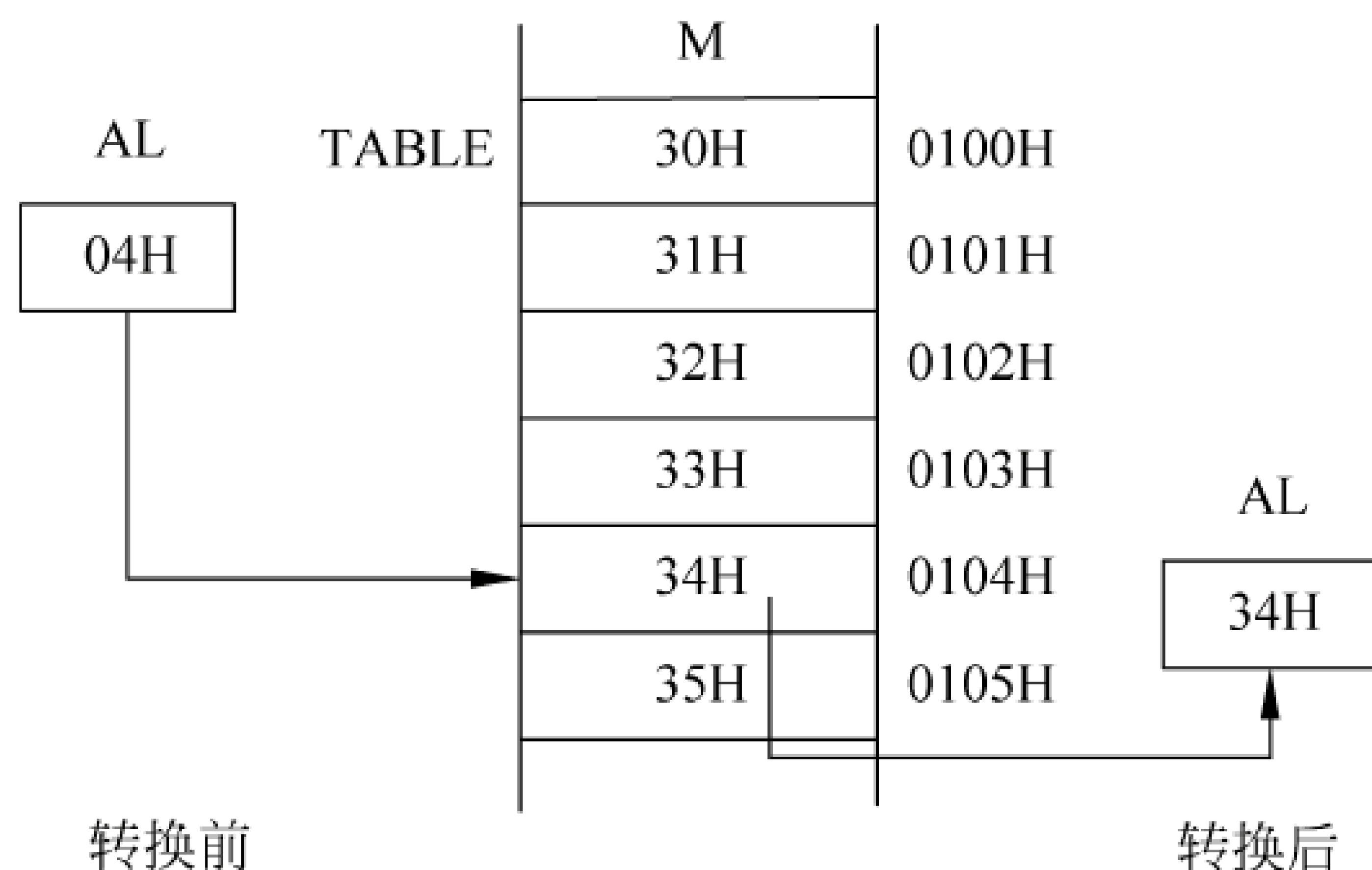


图 5-5 XLAT 指令执行过程示意图

【例 5-6】 以字节变量 TABLE 为首址的 16 个单元中，连续存放 0～15 的平方值(平方表)，任意给一个存放在 X 字节单元中的数 num(0≤num≤15)，例如 num 为 12，查表求 X 的平方值，并把结果存入 Y 字节单元中。

本题可以采用 XLAT 指令查表实现，在数据段中定义好平方表，X 的值实际上就是要转换的代码相对于表格首地址的位移量，通过查表得到相应的平方值。程序如下：

```
DATA    SEGMENT
 TABLE   DB   0,1,4,9,16,25,36,49
         DB   64,81,100,121,144,169,196,225
     X   DB   12
     Y   DB   ?
DATA    ENDS
CODE    SEGMENT
    ASSUME    CS:CODE,DS:DATA
START:   MOV   AX, DATA
         MOV   DS, AX
         MOV   BX, OFFSET TABLE
         MOV   AL, X
         XLAT
         MOV   Y, AL
         MOV   AH, 4CH
         INT   21H
CODE   ENDS
       END   START
```

在集成实验环境中运行，然后通过 DEBUG 逐条指令单步执行，可以查看 XLAT 指令的换码功能。执行 XLAT 指令前，(13C5：0000)开始的存储区存放 16 个数，即平方表，AL

的内容为 0CH，即十进制数 12；执行 XLAT 指令后，AL 的内容为 90H，即十进制数 144，相当于查表求平方值。运行结果如下：

```
-T
AX=13C5  BX=0000  CX=0033  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13C7  IP=0003   NV UP EI PL NZ NA PO NC
13C7:0003 8ED8          MOV     DS,AX
-T
AX=13C5  BX=0000  CX=0033  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C7  IP=0005   NV UP EI PL NZ NA PO NC
13C7:0005 BB0000        MOV     BX,0000
-D DS:0 L10
13C5:0000  00 01 04 09 10 19 24 31-40 51 64 79 90 A9 C4 E1   ......$1@Qdy....
-T
AX=13C5  BX=0000  CX=0033  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C7  IP=0008   NV UP EI PL NZ NA PO NC
13C7:0008 A01000        MOV     AL,[0010]                    DS:0010=0C
-T
AX=130C  BX=0000  CX=0033  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C7  IP=000B   NV UP EI PL NZ NA PO NC
13C7:000B D7            XLAT
-T
AX=1390  BX=0000  CX=0033  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C7  IP=000C   NV UP EI PL NZ NA PO NC
13C7:000C A21100        MOV     [0011],AL                    DS:0011=00
-T
AX=1390  BX=0000  CX=0033  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C7  IP=000F   NV UP EI PL NZ NA PO NC
13C7:000F B44C          MOV     AH,4C
-D DS:0 L12
13C5:0000  00 01 04 09 10 19 24 31-40 51 64 79 90 A9 C4 E1        ......$1@Qdy....
13C5:0010  0C 90                                                  ..
-
```

【例 5-7】 用 XLAT 指令将二进制表示的十六进制数字转换成 ASCII 码显示出来。

由于程序中涉及循环指令和 2 号 DOS 功能调用问题，如果不能理解，可以参阅后续章节的相关知识以后，再学习如下程序：

```
TAB_SEG  SEGMENT
    TAB_DA    DB 30H,31H,32H,33H,34H,35H
              DB 36H,37H, 38H,39H
              DB 41H,42H,43H,44H,45H,46H
    TAB_HEX   DB  0,1,2,3,4,5,6,7,8,9
              DB  0AH,0BH,0CH,0DH, 0EH,0FH
TAB_SEG  ENDS
COSEG  SEGMENT
  ASSUME  CS:COSEG, DS:TAB_SEG
START:MOV  AX,TAB_SEG
      MOV  DS,AX
      MOV  CX,10H
```

```
        MOV   BX,OFFSET   TAB_DA
        MOV   SI,OFFSET   TAB_HEX
NEXT:   MOV   AL,[SI]
        XLAT  TAB_DA            ;换码,得到的 ASCII 码在 AL 中
        MOV   DL,AL             ;将 ASCII 码送入 DL
        MOV   AH,2
        INT   21H               ;显示
        INC   SI
        LOOP  NEXT
        MOV   AH,4CH
        INT   21H
COSEG   ENDS
        END   START
```

5.3 其他传送指令

5.3.1 地址传送指令

1. LEA 指令

格式：LEA REG,SRC

功能：把源操作数(SRC)的有效地址传送至通用寄存器(REG)中。

例如：

```
LEA   BX,VAR              ;BX ←变量 VAR 的位移值
LEA   AX,[BP][DI]         ;AX ←(BP) + (DI)
```

注意 LEA 和 MOV 二者的区别：

```
LEA   BX,VAR              ;等价于 MOV   BX,OFFSET   VAR
MOV   BX,VAR              ;BX ←变量 VAR 的值
```

2. LDS 指令

格式：LDS REG,SRC

功能：双字长的源操作数(SRC)低地址中的字传送至目的寄存器(REG)；高地址中的字传送至 DS。

3. LES 指令

格式：LES REG,SRC

功能：双字长的源操作数(SRC)低地址中的字传送至目的寄存器(REG)；高地址中的字传送至 ES。

	DS段	
TABLE	01H	0100H
	02H	0101H
	03H	0102H
	04H	0103H
	05H	0104H

图 5-6 数据存储情况示意图

【例 5-8】 如图 5-6 所示，TABLE 指向 DS:0100H，理解如下各指令的含义。

```
MOV   BX, WORD PTR TABLE    ;(BX) = 0201H
MOV   BX, OFFSET TABLE      ;(BX) = 0100H
LEA   BX, TABLE             ;(BX) = 0100H
```

根据例 5-8 中给出的已知数据定义和汇编指令，可以编写程序如下：

```
DATA SEGMENT
   ORG 0100H
 TABLE DB 01H,02H,03H,04H,05H
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:  MOV   AX,DATA
        MOV   DS,AX
        MOV   BX,WORD PTR TABLE
        MOV   BX, OFFSET TABLE
        LEA   BX,TABLE
        MOV   AH,4CH
        INT   21H
CODE ENDS
        END   START
```

在集成实验环境中运行，然后通过 DEBUG 逐条指令单步执行，查看 BX 寄存器内容的变化情况，理解 LEA 指令和 MOV 指令的区别。运行情况如下：

```
-T
AX=13C5  BX=0000  CX=0124  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13D6  IP=0003   NV UP EI PL NZ NA PO NC
13D6:0003 8ED8            MOV      DS,AX
-T
AX=13C5  BX=0000  CX=0124  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13D6  IP=0005   NV UP EI PL NZ NA PO NC
13D6:0005 8B1E0001        MOV      BX,[0100]                   DS:0100=0201
-D DS:0100 L5
13C5:0100   01 02 03 04 05                                      .....
-T
AX=13C5  BX=0201  CX=0124  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13D6  IP=0009   NV UP EI PL NZ NA PO NC
13D6:0009 BB0001          MOV      BX,0100
-T
AX=13C5  BX=0100  CX=0124  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13D6  IP=000C   NV UP EI PL NZ NA PO NC
13D6:000C 8D1E0001        LEA      BX,[0100]                     DS:0100=0201
-T
AX=13C5  BX=0100  CX=0124  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13D6  IP=0010   NV UP EI PL NZ NA PO NC
13D6:0010 B44C            MOV      AH,4C
-
```

5.3.2　标志寄存器传送指令

1. 标志送 AH 指令

格式：LAHF

执行操作：(AH) ← (FLAGS 的低字节)

2. AH 送标志寄存器指令

格式：SAHF

执行操作：(FLAGS 的低字节) ←(AH)

3. 标志进栈指令

格式：PUSHF

执行操作：(SP) ←(SP) − 2

((SP)+1, (SP))←(FLAGS)

4. 标志出栈指令

格式：POPF

执行操作：(FLAGS) ←((SP)+1, (SP))

(SP)←(SP) + 2

5.4 实验内容

【实验目的】 通过上机操作，深入理解数据传送指令的使用方法，掌握简单的顺序结构程序设计，能运用 DEBUG 单步运行程序并查看寄存器和存储器内容的变化，掌握基本的程序调试方法。

【实验 5-1】 若(BX)＝2000H，(SI)＝3000H，(DI)＝4000H，(DS)＝1020H，执行下列指令序列后，(AX)＝？(DS:2000)＝？(DS:3000)＝？(DS:4000)＝？要求上机验证所得到的分析结果。

指令序列如下：

```
MOV  AL,2AH
MOV  [BX],AL
MOV  AH,BH
MOV  [SI],AH
MOV  AH,BL
MOV  [DI],AH
MOV  AL,[SI]
```

逐条分析指令，可以得到结果：(AX)＝0020H，(DS:2000)＝ 2AH，(DS:3000)＝20H，(DS:4000)＝ 00H。

上机验证指令的执行结果，可以在 DEBUG 中设置寄存器的初值，然后用 A 命令将给出的指令序列输入，用 T 命令逐条指令单步执行，查看相关寄存器和存储器内容的变化。操作过程和运行结果如下：

```
-R BX
BX 0000
:2000
-R SI
SI 0000
:3000
-R DI
DI 0000
:4000
-R DS
DS 136E
```

```
:1020
-A
136E:0100 MOV AL,2A
136E:0102 MOV [BX],AL
136E:0104 MOV AH,BH
136E:0106 MOV [SI],AH
136E:0108 MOV AH,BL
136E:010A MOV [DI],AH
136E:010C MOV AL,[SI]
136E:010E
-T
AX=002A  BX=2000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=3000  DI=4000
DS=1020  ES=136E  SS=136E  CS=136E  IP=0102   NV UP EI PL NZ NA PO NC
136E:0102 8807          MOV     [BX],AL                         DS:2000=45
-T
AX=002A  BX=2000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=3000  DI=4000
DS=1020  ES=136E  SS=136E  CS=136E  IP=0104   NV UP EI PL NZ NA PO NC
136E:0104 88FC          MOV     AH,BH
-T
AX=202A  BX=2000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=3000  DI=4000
DS=1020  ES=136E  SS=136E  CS=136E  IP=0106   NV UP EI PL NZ NA PO NC
136E:0106 8824          MOV     [SI],AH                         DS:3000=00
-T
AX=202A  BX=2000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=3000  DI=4000
DS=1020  ES=136E  SS=136E  CS=136E  IP=0108   NV UP EI PL NZ NA PO NC
136E:0108 88DC          MOV     AH,BL
-T
AX=002A  BX=2000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=3000  DI=4000
DS=1020  ES=136E  SS=136E  CS=136E  IP=010A   NV UP EI PL NZ NA PO NC
136E:010A 8825          MOV     [DI],AH                         DS:4000=00
-T
AX=002A  BX=2000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=3000  DI=4000
DS=1020  ES=136E  SS=136E  CS=136E  IP=010C   NV UP EI PL NZ NA PO NC
136E:010C 8A04          MOV     AL,[SI]                         DS:3000=20
-T
AX=0020  BX=2000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=3000  DI=4000
DS=1020  ES=136E  SS=136E  CS=136E  IP=010E   NV UP EI PL NZ NA PO NC
136E:010E 0000          ADD     [BX+SI],AL                      DS:5000=00
-D DS:2000 L1
1020:2000  2A                                                  *
-D DS:3000 L1
1020:3000  20
-D DS:4000 L1
1020:4000  00                                                  .
-
```

【实验 5-2】 设数据段定义如下：

```
DATA SEGMENT
    BUF1  DB 25H,7,8
    BUF2  DW 1A2BH,357H
```

```
    BUF3   DB 10 DUP ('ABC')
    BUF4   DW $ - BUF3
    BUF5   DB 20 DUP(0)
DATA ENDS
```

写出下列指令的运行结果，并上机编写汇编语言程序验证分析得到的结果。

(1) 执行“MOV AX, WORD PTR BUF1”后，(AX)=？

(2) 执行“LEA BX, BUF3”后，(BX)=？

(3) 执行“MOV AL, BUF3+2”后，(AL)=？

(4) 执行“MOV CX, BUF4”后，(CX)=？

(5) 执行“MOV AX, BUF2+1”后，(AX)=？

(6) 执行“MOV SI, OFFSET BUF5”后，(SI)=？

分析指令的执行结果，首先要搞清楚数据段的存储情况，画出存储器示意图，如图 5-7 所示，然后再逐条分析指令执行情况。具体情况如下：

(1) 虽然 BUF1 是按字节变量定义，但是也可以通过类型转换操作符按字属性读取数据，所以指令执行后(AX)=0725H。

(2) 执行“LEA BX, BUF3”后，由于 BUF3 距离数据段首地址的位移量是 7，因此指令执行后(BX)=0007H。

(3) 字符存储是按 ASCII 码值存储，(BUF3+2)单元是字符'C'的 ASCII 码值，所以指令执行后(AL)=43H。

(4) \$是地址计数器的当前值，\$－BUF3 实际上就是 30 个字符存储所占的字节数 30，所以指令执行后(CX)=001EH。

(5) 由于 AX 是 16 位寄存器，因此该指令执行时按字属性读取(BUF2+1)单元的数据，低位字节在(BUF2+1)单元，高位字节在(BUF2+2)单元，因此(AX)=571AH。

(6) BUF5 距离数据段首地址的位移量是 39，所以指令执行后(SI)=0027H。

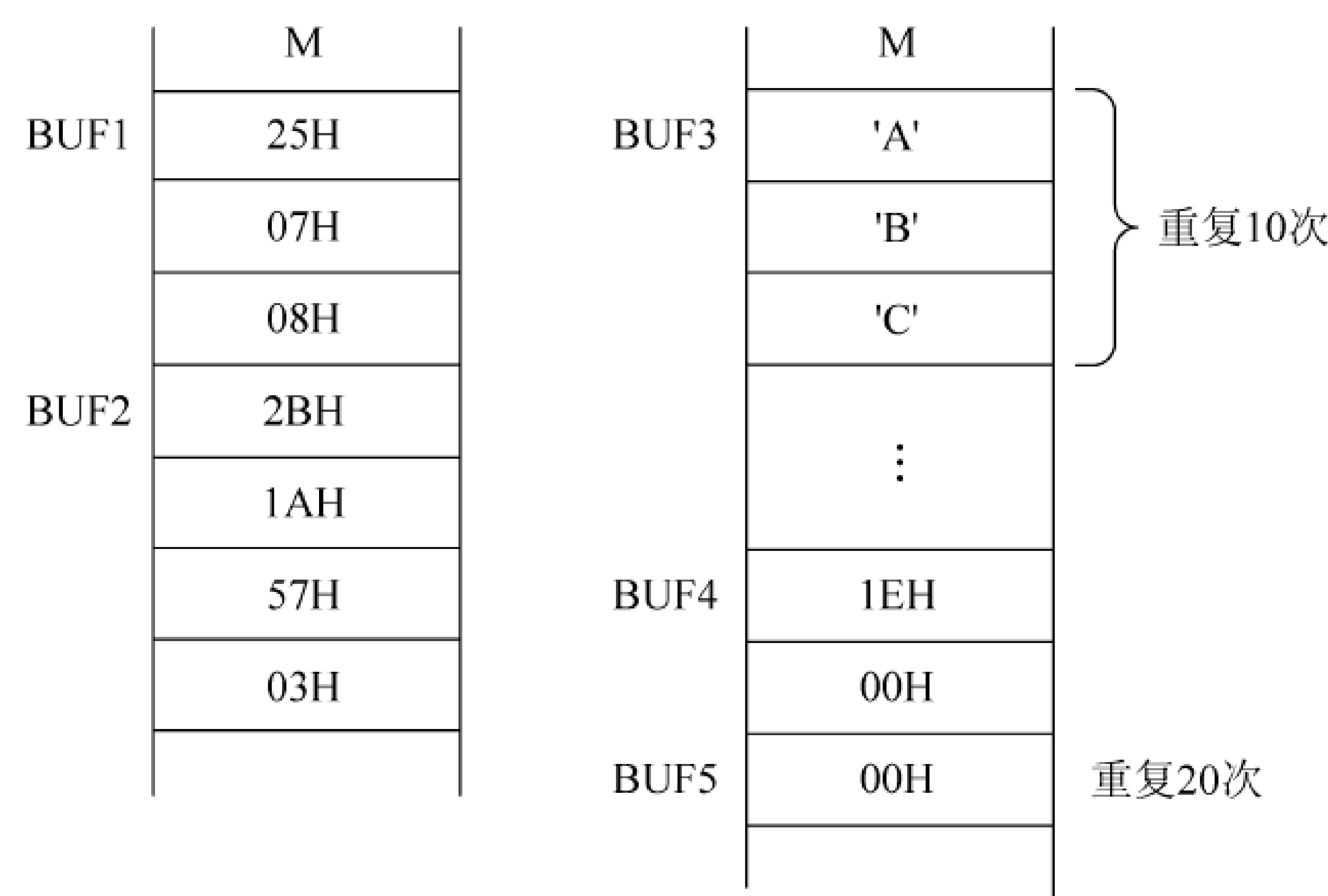


图 5-7 实验 5-2 数据段存储情况示意图

如果上机验证分析结果，可以编写汇编语言程序如下：

```
DATA SEGMENT
```

```
        BUF1   DB 25H,7,8
        BUF2   DW 1A2BH,357H
        BUF3   DB 10 DUP ('ABC')
        BUF4   DW $ - BUF3
        BUF5   DB 20 DUP(0)
    DATA ENDS
    CODE SEGMENT
        ASSUME CS:CODE,DS:DATA
    START: MOV AX,DATA
           MOV DS,AX
           MOV AX, WORD PTR BUF1
           LEA BX, BUF3
           MOV AL, BUF3 + 2
           MOV CX, BUF4
           MOV AX, BUF2 + 1
           MOV SI, OFFSET BUF5
           MOV AH,4CH
           INT 21H
    CODE ENDS
           END START
```

在集成实验环境中，单步运行，数据段装填语句执行完，可以查看存储器中的数据存储情况。运行结果如下：

```
-T
AX = 13C5  BX = 0000  CX = 005D  DX = 0000  SP = 0000  BP = 0000  SI = 0000  DI = 0000
DS = 13B5  ES = 13B5  SS = 13C5  CS = 13C9  IP = 0003   NV UP EI PL NZ NA PO NC
13C9:0003 8ED8          MOV     DS,AX
-T
AX = 13C5  BX = 0000  CX = 005D  DX = 0000  SP = 0000  BP = 0000  SI = 0000  DI = 0000
DS = 13C5  ES = 13B5  SS = 13C5  CS = 13C9  IP = 0005   NV UP EI PL NZ NA PO NC
13C9:0005 A10000        MOV     AX,[0000]                  DS:0000 = 0725
-D DS:0
13C5:0000  25 07 08 2B 1A 57 03 41 - 42 43 41 42 43 41 42 43   %..+.W.ABCABCABC
13C5:0010  41 42 43 41 42 43 41 42 - 43 41 42 43 41 42 43 41   ABCABCABCABCABCA
13C5:0020  42 43 41 42 43 1E 00 00 - 00 00 00 00 00 00 00 00   BCABC...........
13C5:0030  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00   ................
13C5:0040  B8 C5 13 8E D8 A1 00 00 - 8D 1E 07 00 A0 09 00 8B   ................
13C5:0050  0E 25 00 A1 04 00 BE 27 - 00 B4 4C CD 21 00 00 00   .%.....'..L.!...
13C5:0060  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00   ................
-
```

从运行结果看，与图 5-7 给出的结果一致。继续用 T 命令单步运行，可以看到每条指令执行后的寄存器内容变化。运行结果如下：

```
-T
AX = 0725  BX = 0000  CX = 005D  DX = 0000  SP = 0000  BP = 0000  SI = 0000  DI = 0000
DS = 13C5  ES = 13B5  SS = 13C5  CS = 13C9  IP = 0008   NV UP EI PL NZ NA PO NC
13C9:0008 8D1E0700      LEA     BX,[0007]                  DS:0007 = 4241
-T
```

```
AX = 0725   BX = 0007   CX = 005D   DX = 0000   SP = 0000   BP = 0000   SI = 0000   DI = 0000
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C9   IP = 000C    NV UP EI PL NZ NA PO NC
13C9:000C A00900            MOV       AL,[0009]                    DS:0009 = 43
-T
AX = 0743   BX = 0007   CX = 005D   DX = 0000   SP = 0000   BP = 0000   SI = 0000   DI = 0000
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C9   IP = 000F    NV UP EI PL NZ NA PO NC
13C9:000F 8B0E2500          MOV       CX,[0025]                    DS:0025 = 001E
-T
AX = 0743   BX = 0007   CX = 001E   DX = 0000   SP = 0000   BP = 0000   SI = 0000   DI = 0000
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C9   IP = 0013    NV UP EI PL NZ NA PO NC
13C9:0013 A10400            MOV       AX,[0004]                    DS:0004 = 571A
-T
AX = 571A   BX = 0007   CX = 001E   DX = 0000   SP = 0000   BP = 0000   SI = 0000   DI = 0000
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C9   IP = 0016    NV UP EI PL NZ NA PO NC
13C9:0016 BE2700            MOV       SI,0027
-T
AX = 571A   BX = 0007   CX = 001E   DX = 0000   SP = 0000   BP = 0000   SI = 0027   DI = 0000
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C9   IP = 0019    NV UP EI PL NZ NA PO NC
13C9:0019 B44C              MOV       AH,4C
-
```

通过上机实验,可以加深对汇编指令及存储器操作数的理解,有效地提高学生的应用实践能力。

习　　题

1. 根据要求写出相应的汇编语言指令或指令序列:

(1) 立即数 17 送入 CL 寄存器。

(2) 寄存器 BX 的内容传送给 AX。

(3) 将 AX 寄存器的内容传送到字变量 BUFFER 单元。

(4) 功能与"LEA BX, BLOCK"等价的指令。

(5) 以 BX 和 SI 寄存器作基址变址寻址方式,把该单元中的一个字传送到 AX。

(6) 利用 PUSH/POP 指令实现寄存器 SI 和 DI 的内容互换。

(7) 将 BUFFER1 的一个字节数据传送到 BUFFER2 字节单元中。

2. 若(SP)=2017H,(AX)=3040H,(BX)=5060H,则:

(1) 执行"PUSH　AX"后,(SP)=?

(2) 再执行"PUSH　BX"及"POP　AX"两条指令后,(SP)=? (AX)=? (BX)=?

3. 写出下列指令序列执行后的结果:

(1)
```
MOV   AX,2017H
MOV   BX,AX
```

执行后(AX)=? (BX)=?

(2)
```
MOV   AX,1234H
MOV   BX,5678H
MOV   [BX],AL
MOV   AH,BL
```

执行后(AX)=?(BX)=?

(3)
```
MOV AX,2000H
MOV BX,3000H
XCHG AX,BX
```

执行后(AX)=?(BX)=?

4. 数据段定义如下:

```
DATAS SEGMENT
  A1   DB 10 DUP (?)
  A2   DW 1234H,5678H,9ABCH,0DEFH
  A3   DW $ - A2
  A4   DW A2
  A5   DB 6,5,2
DATAS ENDS
```

分别完成下列问题,并编写程序验证分析结果。

(1) 用一条指令将 A2 的偏移地址送 BX;

(2) 将字符'A'存入变量 A1 的第 6 个字节单元中;

(3) 将 A2 的第 3 个字节的内容送给 AL;

(4) 将 A2 的第 3 个单元开始的字数据送给 AX。

第 6 章

算术运算程序

本章主要讲述算术运算指令及其在程序设计中的应用，重点是加法、减法、乘法和除法指令的功能以及十进制调整指令的用法，同时介绍算术运算程序设计方法。

6.1 算术运算概述

在算术运算指令中有两种类型的数据：无符号二进制数和带符号二进制数。对于无符号二进制数，所有数位都看成数据位。8 位无符号数表示的数为 0～255，16 位无符号数表示的数为 0～65 535；而带符号二进制数，最高位是符号位；数据位用补码表示。因此，8 位带符号数表示的数为－128～＋127，16 位带符号数表示的数为－32 768～＋32 767。

在十进制数运算中，使用 BCD 码，分为压缩型 BCD 码和非压缩型 BCD 码两种格式。压缩型 BCD 码，用 4 位二进制数表示一位十进制数，一个字节存放两个 BCD 码，高位 BCD 码比低位 BCD 码权值大；非压缩型 BCD 码，8 位为一组表示一个十进制数，BCD 码存放在字节的低 4 位，高 4 位无意义。

算术运算与数据格式及状态标志密切相关，涉及的标志位有：进位标志 CF、符号标志 SF、零标志 ZF、溢出标志 OF 等。

8086 提供二进制运算和十进制运算指令，可以完成加、减、乘、除运算，可以是 8 位或 16 位。算术运算指令包括：

- 加法指令：ADD、ADC、INC；
- 减法指令：SUB、SBB、DEC、NEG、CMP；
- 乘法指令：MUL、IMUL；
- 除法指令：DIV、IDIV；
- 十进制调整指令：DAA、DAS、AAA、AAS、AAM、AAD。

6.2 二进制数的算术运算

6.2.1 加法运算

1. 加法指令

（1）加法指令：ADD DST, SRC

执行操作：(DST) ← (DST) ＋ (SRC)

（2）带进位加法指令：ADC DST, SRC

执行操作：(DST)←(DST)+(SRC)+CF

其中，DST 代表目的操作数，SRC 代表源操作数，即两个操作数相加并将结果存放在目的操作数中。

2. 加法指令应用

(1) 寄存器→寄存器。

```
ADD   AX,BX
ADD   DH,DL
```

(2) 存储器→寄存器。

```
ADD   AX,[BX][SI]
ADD   AL,DATA[DI]
```

(3) 寄存器→存储器。

```
ADD   DETA[DI],BX
ADD   [BX],DX
```

(4) 立即数→寄存器，存储器。

```
ADD   AX,456
ADD   DATA[DI],50H
```

注意：不能从存储器加到存储器，段寄存器也不能作为操作数。使用寄存器作为目的操作数时，尽量使用累加器，这样目标代码短。加法指令的执行结果全面影响状态标志位。其中，CF 位表示进位/借位，也可以表示无符号数相加的溢出；OF 位表示带符号数相加的溢出。

【例 6-1】 下列指令执行后，AL=？ BL=？ CF=？

```
MOV AL,17H
MOV BL,23H
ADD AL, BL
```

前两条指令相当于给寄存器赋值，即 AL=17H，BL=23H，执行加法指令时，进行二进制的加法：

```
      00010111AL
+)    00100011BL
------------------
      00111010AL
```

指令执行后，AL=3AH，BL=23H，CF=0。在 DEBUG 下运行汇编指令，结果如下：

```
-A
136E:0100 MOV AL,17
136E:0102 MOV BL,23
136E:0104 ADD AL,BL
136E:0106
-T
AX=0017   BX=0000   CX=0000   DX=0000   SP=FFEE   BP=0000   SI=0000   DI=0000
DS=136E   ES=136E   SS=136E   CS=136E   IP=0102    NV UP EI PL NZ NA PO NC
```

```
136E:0102 B323          MOV     BL,23
-T
AX=0017  BX=0023  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0104   NV UP EI PL NZ NA PO NC
136E:0104 00D8          ADD     AL,BL
-T
AX=003A  BX=0023  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0106   NV UP EI PL NZ NA PE NC
136E:0106 0000          ADD     [BX+SI],AL                    DS:0023=FF
-
```

【例 6-2】 多字节加法。设目的操作数存放在 DX(高位)和 AX(低位)中,源操作数在 BX (高位)和 CX 中,执行前:(DX) = 0012H,(AX) = 8364H,(BX) = 0005H,(CX) = 0F213H。则执行下列指令序列后,运算结果是什么? 对状态标志位有何影响?

```
ADD  AX, CX
ADC  DX, BX
```

(1) 执行“ADD AX, CX”后,(AX) = 7577H;标志位:CF=1,OF=1,SF=0,ZF=0;

(2) 执行“ADC DX, BX”后,(DX) = 0018H;标志位:CF=0,OF=0,SF=0,ZF=0。

利用 DEBUG 上机验证时,先用 R 命令设置好各寄存器的初值,然后输入汇编指令,单步执行,部分操作运行结果如下:

```
-R
AX=8364  BX=0005  CX=F213  DX=0012  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0100   NV UP EI PL NZ NA PO NC
136E:0100 0000          ADD     [BX+SI],AL                    DS:0005=9A
-A
136E:0100 ADD AX,CX
136E:0102 ADC DX,BX
136E:0104
-T
AX=7577  BX=0005  CX=F213  DX=0012  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0102   OV UP EI PL NZ NA PE CY
136E:0102 11DA          ADC     DX,BX
-T
AX=7577  BX=0005  CX=F213  DX=0018  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0104   NV UP EI PL NZ NA PE NC
136E:0104 0000          ADD     [BX+SI],AL                    DS:0005=9A
-
```

3. 加 1 指令

指令格式:INC OPR

执行操作:(OPR) ←(OPR) + 1

说明:该指令是单操作数指令,OPR 代表操作数。当寻址方式为存储器操作数时,必须加类型说明符。

注意:该指令不影响 CF 标志位,但对其他标志位有影响。一般用于循环程序中修改地址或循环次数。

6.2.2 减法运算

减法指令格式：SUB DST, SRC

执行操作：(DST) ←(DST) － (SRC)

带借位减法指令格式：SBB DST, SRC

执行操作：(DST) ←(DST) － (SRC) － CF

减 1 指令格式：DEC OPR

执行操作：(OPR) ←(OPR) － 1

求补指令格式：NEG OPR

执行操作：(OPR) ← － (OPR)

比较指令格式：CMP OPR1, OPR2

执行操作：(OPR1) － (OPR2)

注意：指令中的操作数与加法指令相同，除 DEC 指令不影响 CF 标志外，均对条件标志位有影响。

【例 6-3】 若 AL＝65H，BL＝03H，CF＝1，则执行指令“SBB AL，BL”后，结果是什么？标志位 CF＝？

在计算机内部，减法实际上是通过加法来完成的，即先把源操作数(SRC)变为补码，标志位 CF 也变为补码，然后再做加法。执行“SBB AL，BL”指令的算式为：

```
      01100101   AL
      11111101   BL(－3 的补码)
  ＋) 11111111   CF(－1 的补码)
  -----------------
      01100001   AL
```

指令执行后，AL＝61H，BL＝03H，CF＝0。在 DEBUG 中验证时，可以先用 R 命令设置好寄存器和标志位的初值，然后运行汇编指令，操作过程和运行结果如下：

```
-R AX
AX 0000
:0065
-R BX
BX 0000
:0003
-R F
NV UP EI PL NZ NA PO NC   -CY
-A
136E:0100 SBB AL,BL
136E:0102
-T
AX=0061  BX=0003  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0102   NV UP EI PL NZ NA PO NC
136E:0102 0000          ADD     [BX+SI],AL                    DS:0003=9F
-
```

【例 6-4】 两个无符号双字的减法。设被减数存放在 DX、AX 中，其中 DX 存放高位字；减数存放在 BX、CX 中，其中 BX 存放高位字。指令执行前 DX＝0036H，AX＝7546H，

BX＝0012H，CX＝8427H。执行如下指令序列后，结果如何？

```
SUB   AX,CX
SBB   DX,BX
```

（1）执行“SUB　AX,CX”后，AX＝0F11FH，CX＝8247H，CF＝1，OF＝1，SF＝1，ZF＝0；

（2）执行“SBB　DX,BX”后，DX＝0023H，BX＝0012H，CF＝0，OF＝0，SF＝0，ZF＝0。

利用 DEBUG 上机验证时，先用 R 命令设置好各寄存器的初值，然后单步执行汇编指令，运行结果如下：

```
-R
AX=7546  BX=0012  CX=8427  DX=0036  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0100   NV UP EI PL NZ NA PO NC
136E:0100 0000          ADD     [BX+SI],AL                 DS:0012=17
-A
136E:0100 SUB AX,CX
136E:0102 SBB DX,BX
136E:0104
-T
AX=F11F  BX=0012  CX=8427  DX=0036  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0102   OV UP EI NG NZ AC PO CY
136E:0102 19DA          SBB     DX,BX
-T
AX=F11F  BX=0012  CX=8427  DX=0023  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0104   NV UP EI PL NZ NA PO NC
136E:0104 0000          ADD     [BX+SI],AL                 DS:0012=17
-
```

注意：使用 ADC 和 SBB 指令时，必须确认自上次加法/减法之后，进位标志 CF 没有被更改过，否则 CF 状态将丢失。

例如：如下代码不能正确实现 CX:BX 与 DX:AX 相加。

```
ADD   AX,BX
SUB   SI,SI             ;SI←0,清除标志位
ADC   DX,CX             ;该指令不能正常工作,因为 ADD 指令产生的标志位被破坏
```

【例 6-5】　x、y、z 均为双精度数，分别存放在地址为 X，X+2；Y，Y+2；Z，Z+2 的存储单元中，用指令序列实现 w ←x ＋ y ＋ 88 － z，并用 W，W+2 单元存放 w。

双精度数在数据定义时占 4B，其算术运算是多字节数的加法/减法。指令序列如下：

```
MOV   AX,  X
MOV   DX,  X+2
ADD   AX,  Y
ADC   DX,  Y+2          ;x+y
ADD   AX,  88
ADC   DX,  0            ;x+y+88
SUB   AX,  Z
SBB   DX,  Z+2          ;x+y+88-z
MOV   W,   AX
MOV   W+2, DX           ;结果存入 W, W+2 单元
```

对于例 6-5，假设数据段用 DW 伪指令定义 x、y、z 的值分别为 12345678H、7694ABCDH、009400CDH，则可以编写相应的汇编语言程序。

（1）用直接寻址方式实现。

```
DATA SEGMENT
 X  DW  5678H,1234H
 Y  DW  0ABCDH,7694H
 Z  DW  00CDH,0094H
 W  DW  ?,?
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:MOV   AX,DATA
      MOV   DS,AX
      MOV   AX,X
      MOV   DX,X+2
      ADD   AX,Y
      ADC   DX,Y+2          ;x+y
      ADD   AX,88
      ADC   DX,0            ;x+y+88
      SUB   AX,Z
      SBB   DX,Z+2          ;x+y+88-z
      MOV   W,AX
      MOV   W+2,DX          ;结果存入 W, W+2 单元
      MOV   AH,4CH
      INT   21H
CODE ENDS
    END START
```

在 DEBUG 下，单步运行，查看程序的运行结果如下：

```
-D DS:0
13C5:0000  78 56 34 12 CD AB 94 76-CD 00 94 00 D0 01 35 88   xV4....v......5.
13C5:0010  B8 C5 13 8E D8 A1 00 00-8B 16 02 00 03 06 04 00   ................
13C5:0020  13 16 06 00 83 C0 58 83-D2 00 2B 06 08 00 1B 16   ......X...+.....
13C5:0030  0A 00 A3 0C 00 89 16 0E-00 B4 4C CD 21 00 00 00   ..........L.!...
13C5:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

（2）采用寄存器相对寻址实现。

```
DATA SEGMENT
 X  DW  5678H,1234H
 Y  DW  0ABCDH,7694H
 Z  DW  00CDH,0094H
 W  DW  ?,?
DATA ENDS
CODE SEGMENT
```

```
    ASSUME CS:CODE,DS:DATA
START:MOV   AX,DATA
      MOV   DS,AX
      MOV   BX,OFFSET X
      MOV   AX,[BX]
      MOV   DX,2[BX]        ;X值送给DX:AX
      ADD   AX,4[BX]
      ADC   DX, 6[BX]       ;X+Y
      ADD   AX, 88
      ADC   DX, 0           ;X+Y+88
      SUB   AX, 8[BX]
      SBB   DX,  10[BX]     ;X+Y+88-Z
      MOV   12[BX],AX
      MOV   14[BX],DX       ;结果存入W,W+2单元
      MOV   AH,4CH
      INT   21H
CODE ENDS
    END START
```

当然,对于该程序还可以采用其他寻址方式,如寄存器间接寻址等,请读者自行完成。

6.2.3 乘法运算

无符号数乘法指令格式:MUL SRC

带符号数乘法指令格式:IMUL SRC

乘法运算指令分为以下几种情形:

1. 字节操作数(8×8):(AX) ← (AL) * (SRC)

因子之一在AL中,另一个因子SRC为8位通用寄存器或存储器操作数,16位乘积在AX中。

例如:
```
MOV  AL,25
MOV  DH,40
MUL  DH              ;25×40=1000
```

例如:字节变量A和B进行无符号乘,结果放在字变量C里。

```
MOV  AL,  A
MUL  B
MOV  C,  AX
```

2. 字操作数(16×16):(DX, AX) ←(AX) * (SRC)

因子之一在AX中,另一个因子SRC为16位通用寄存器或存储器操作数,32位乘积在DX:AX中(高16位在DX中,低16位在AX中)。

例如:
```
MOV AX,1000
MUL AX               ;求平方,结果在DX:AX中
```

6.2.4 除法运算

无符号数除法指令格式:DIV SRC

带符号数除法指令格式:IDIV SRC

除法运算指令分为以下几种情形：

1. 16 位÷8 位

入口：被除数在 AX 中，8 位操作数 SRC 作为除数，可以是存储器变量或 8 位通用寄存器。

出口：8 位的商在 AL 中，8 位的余数在 AH 中。

例如：
```
MOV AX,51
MOV DL,10
DIV DL               ;(AL) = 05H,(AH) = 01H
```

注意：除法结果不能太大，太大（商>255）时产生 0 号中断，关于中断的相关内容，参见第 12 章。

2. 32 位÷16 位

入口：被除数在 DX:AX 中，16 位操作数 SRC 作为除数，可以是存储器变量或 16 位通用寄存器。

出口：16 位的商在 AX 中，16 位的余数在 DX 中。

例如：
```
MOV AX,2
MOV DX,1             ;(DX:AX) = 00010002H
MOV BX,10H
DIV BX               ;(AX) = 1000H,   (DX) = 0002H
```

注意：除法结果不能太大，太大（商>65 535）时产生 0 号中断。

【例 6-6】 对两个字节无符号数求其平均值。它们分别存放在 X 和 Y 字节存储单元中，而平均值存放在 Z 字节存储单元中，试编写完整的程序。

为方便编程，不妨分别给 X 和 Y 一个具体值。程序如下：

```
DATA     SEGMENT
  X    DB    88H
  Y    DB    8CH
  Z    DB     ?
DATA     ENDS
CODE   SEGMENT
      ASSUME   CS:CODE,DS:DATA
START:  MOV    AX,DATA
        MOV    DS,AX           ;设置数据段的段基值
        MOV    AL,X            ;取第一个数
        ADD    AL,Y            ;两数相加
        MOV    AH,0
        ADC    AH,0            ;回收进位值
        MOV    BL,2
        DIV    BL              ;AX/2,结果: AL←商,AH←余数
        MOV    Z ,AL           ;商存入 Z 单元中
        MOV    AH,4CH
        INT   21H
CODE   ENDS
  END START
```

在 DEBUG 下，单步运行，查看程序的运行结果如下：

```
-D DS:0
13C5:0000   88 8C 8A 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

```
13C5:0010  B8 C5 13 8E D8 A0 00 00-02 06 01 00 B4 00 80 D4   ................
13C5:0020  00 B3 02 F6 F3 A2 02 00-B4 4C CD 21 00 00 00 00   .........L.!....
13C5:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

6.2.5 符号扩展指令

为了配合带符号数除法指令 IDIV 对被除数的要求，IBM-PC 指令系统专门提供了符号位扩展指令：CBW、CWD。这两个都是无操作数指令，隐含对 AL 或 AX 进行符号扩展，不影响条件标志位。

1. CBW(Convert Byte to Word)

格式：`CBW                ;AL→AX`

执行操作：若(AL)的最高有效位为 0，则(AH)＝ 00H；若(AL)的最高有效位为 1，则(AH)＝ FFH。

2. CWD(Convert Word to Double word)

格式：`CWD                ;AX →(DX, AX)`

执行操作：若(AX)的最高有效位为 0，则(DX)＝ 0000H；若(AX)的最高有效位为 1，则(DX)＝ FFFFH。

【例 6-7】 X、Y、Z、V 均为 16 位带符号数，计算表达式(V －(X * Y ＋Z － 888))/X 的值。

假设 X、Y、Z、V 分别存储在 XX、YY、ZZ、VV 单元，指令序列如下：

```
MOV   AX, XX
IMUL  YY                ;X * Y →(DX,AX)
MOV   CX, AX
MOV   BX, DX
MOV   AX, ZZ
CWD                     ;Z →(DX,AX)
ADD   CX, AX
ADC   BX, DX            ;X * Y + Z →(BX,CX)
SUB   CX, 888
SBB   BX, 0             ;X * Y + Z - 888
MOV   AX, VV
CWD                     ;V →(DX,AX)
SUB   AX, CX
SBB   DX, BX            ;V - ( X * Y + Z - 888)
IDIV  XX                ;(V - ( X * Y + Z - 888))/X→(AX),余数→(DX)
```

6.3 十进制数的算术运算

前面介绍的算术运算指令都是二进制数的运算，为了处理通常的十进制数，指令系统中提供了一组十进制调整指令，即在二进制运算的基础上，加一条十进制调整指令来实现十进

制数的算术运算。

6.3.1　压缩的 BCD 码调整指令

1. 加法的十进制调整指令

格式：DAA　　　　　　　;隐含操作数为 AL,用于 ADD/ADC 之后

功能：对压缩 BCD 码加法结果进行调整。

【例 6-8】 计算 28+49=77。

```
MOV   AL,28H           ;28H 代表 28 的 2 位 BCD 码
ADD   AL,49H           ;49H 代表 49 的 2 位 BCD 码
DAA                    ;AL = 77H
```

ADD 指令进行的加法，是二进制数的加法，得到的和 AL=71H，不是压缩的 BCD 码。执行加法后，标志位 AF=1，使 AL 的低 4 位满足调整的条件。经过 DAA 调整后，得到 AL=77H，是压缩的 BCD 码。DEBUG 下的运行结果如下：

```
-A
136E:0100 MOV AL,28
136E:0102 ADD AL,49
136E:0104 DAA
136E:0105
-T
AX = 0028  BX = 0000  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 136E  ES = 136E  SS = 136E  CS = 136E  IP = 0102   NV UP EI PL NZ NA PO NC
136E:0102 0449             ADD      AL,49
-T
AX = 0071  BX = 0000  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 136E  ES = 136E  SS = 136E  CS = 136E  IP = 0104   NV UP EI PL NZ AC PE NC
136E:0104 27               DAA
-T
AX = 0077  BX = 0000  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 136E  ES = 136E  SS = 136E  CS = 136E  IP = 0105   NV UP EI PL NZ AC PE NC
136E:0105 0000             ADD      [BX + SI],AL                   DS:0000 = CD
-
```

2. 减法的十进制调整指令

格式：DAS　　　　　　　;隐含操作数为 AL,用于 SUB/SBB 之后

功能：对压缩 BCD 码减法结果进行调整。

【例 6-9】 计算 45-18=27。

```
MOV   AL,45H
SUB   AL,18H           ;AL = 2DH
DAS                    ;AL = 27H
```

执行 SUB 后，AL=2DH，不是压缩 BCD 码；执行 DAS 后，AL=27H，是压缩 BCD 码，代表十进制数 27。DEBUG 下的运行结果如下：

```
-A
136E:0100 MOV AL,45
```

```
136E:0102 SUB AL,18
136E:0104 DAS
136E:0105
-T
AX=0045  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0102   NV UP EI PL NZ NA PO NC
136E:0102 2C18          SUB     AL,18
-T
AX=002D  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0104   NV UP EI PL NZ AC PE NC
136E:0104 2F            DAS
-T
AX=0027  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0105   NV UP EI PL NZ AC PE NC
136E:0105 0000          ADD     [BX+SI],AL                    DS:0000=CD
-
```

6.3.2 非压缩的 BCD 码调整指令

非压缩的 BCD 码调整指令相对复杂，使用时注意加法、减法、乘法的非压缩 BCD 码的调整指令都是用在加法、减法、乘法指令之后，而除法的非压缩 BCD 码的调整指令则用在除法指令之前。

1. 加法的 ASCII 码调整指令

格式：AAA

功能：若 AL 的低 4 位大于 9 或 AF=1，则将 AL 的内容加 6，AH 的内容加 1，并将 AF、CF 标志位置 1，将 AL 的高 4 位清 0，截取 AL 内容的低 4 位；否则，只截取 AL 内容的低 4 位。

2. 减法的 ASCII 码调整指令

格式：AAS

功能：若 AL 的低 4 位大于 9 或 AF=1，则将 AL 的内容减去 6，AH 的内容减 1，并将 AF、CF 标志位置 1，将 AL 的高 4 位清 0，截取 AL 内容的低 4 位；否则，只截取 AL 内容的低 4 位。

3. 乘法的 ASCII 码调整指令

格式：AAM

功能：将 AL 中的二进制乘积分解为两位十进制数(BCD 码)，十位数的 BCD 码送 AH，个位数的 BCD 码送 AL。具体操作是将 AL 的内容除以 0AH，商送至 AH，余数送至 AL。

4. 除法的 ASCII 码调整指令

格式：AAD

功能：将 AX 中的两位非压缩型 BCD 码转换成真正的二进制数。具体操作是：AL←AH * 0AH+AL；AH←0。

6.4 实验内容

【实验目的】 通过上机实践，深入理解算术运算指令的功能及其对标志位的影响，掌握算术运算指令在程序设计中的应用。

【实验 6-1】 算术运算指令。分别完成下列问题,并上机验证分析结果。

(1)
```
MOV   AX, 0A95BH
ADD   AX, 8CA2H
```

① 执行后,AX=________,CF=________

② 若将 ADD 改为 SUB,执行后结果 AX=________,CF=________

(2)
```
MOV AL,0FFH
ADD AL,1            ;执行后,AL = ________
```

(3)
```
MOV   AL,80H
SUB    AL,1         ;执行后,AL = ________,OF = ________
```

(4)
```
MOV AX,0AH
MOV CL,6
DIV CL              ;执行后,AX = ________
```

(5)
```
MOV AL,13H
ADD AL,99H
DAA                 ;执行后,AL = ________,CF = ________
```

(6)
```
MOV   AL,96H
MOV   BL, 12H
MUL   BL            ;AX = ________,CF  = ________,OF = ________
```

【实验 6-2】 编写完成下列功能的程序,并利用集成实验环境调试通过,说明如何查看 DATA3 存储区内容的变化。

(1) 在数据段中定义,字变量 DATA1 有两个数据:2D56H 和 7215H;字变量 DATA2 有两个数据:0B678H 和 25A7H;字变量 DATA3 为两个空单元。

(2) 将 DATA1 的第一个字数据传送给 AX。

(3) 将 AX 的内容与 DATA2 的第一个字数据相减,结果存入 DATA3 的第一个空单元。

(4) 将 DATA1 的第二个字数据传送给 AX。

(5) 将 AX 的内容与 DATA2 的第二个字数据相减(要考虑上一次减法的借位),结果存入 DATA3 的第二个空单元。

参考程序:

```
DATAS SEGMENT
  DATA1 DW 2D56H,7215H
  DATA2 DW 0B678H,25A7H
  DATA3 DW ?,?
DATAS ENDS
CODES SEGMENT
    ASSUME CS:CODES,DS:DATAS
START:
    MOV AX,DATAS
    MOV DS,AX
    MOV AX,DATA1
    SUB AX,DATA2
    MOV DATA3,AX
    MOV AX,DATA1 + 2
    SBB AX,DATA2 + 2
```

```
    MOV DATA3 + 2,AX
    MOV AH,4CH
    INT 21H
CODES ENDS
    END START
```

习　　题

1. 分析下列指令的运行结果并通过上机验证。

```
(1) MOV   AL,96H
    MOV   BL, 12H
    IMUL  BL              ;AX = ?  CF  = ?  OF = ?
(2) MOV   AX, 916EH
    MOV   BX, 18F2H
    MUL   BX              ;DX = ?  AX = ?  CF  = ?  OF = ?
(3) MOV DX,0023H
    MOV AX,7546H
    MOV BX,0012H
    MOV CX,9428H
    SUB AX,CX             ;AX = ?  CF  = ?
    SBB DX,BX             ;DX = ?  OF = ?  CF  = ?
(4) MOV AX,689AH
    CMP AH,AL             ;AH = ?  OF = ?  SF = ?  ZF = ?  CF  = ?
(5) MOV AL, - 128
    SUB AL,1              ;AL = ?  OF = ?
(6) MOV AL,13H
    ADD AL,99H
    DAA                   ;AL = ?  CF = ?
```

2. 执行下列指令后,AX 寄存器的内容是什么?

```
TABLE1   DW  1007H,1234H,6517H
ENTRY    DW  3
…
MOV   BX,OFFSET TABLE1
ADD   BX,ENTRY
MOV   AX,[BX]
```

3. 分析下列程序段,说明程序的功能。

```
DATA SEGMENT
    DA1    DB 29H,38H
    DA2    DB 46H,57H
    DA3    DB ?,?
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
```

```
        MOV AL,DA1
        ADD AL,DA2
        DAA
        MOV DA3,AL
        MOV AL,DA1 + 1
        ADC AL,DA2 + 1
        DAA
        MOV DA3 + 1,AL
        MOV AH,4CH
        INT 21H
CODE ENDS
        END START
```

4. 编写程序段,实现单字节数 X 乘以 10。

5. 在 BLOCK 开始的连续四个字节存储单元中,存放有 4 个无符号数。试编程实现将前三个数求和;再减去第四个数,结果存入 RESULT 单元中。(为简化起见,假设运算结果仍为单字节数,运算过程中不考虑有无进位/借位问题。)

6. 将例 6-7 中的程序改写成完整的程序,数据段自行定义,要求上机调试通过,并说明如何验证运行结果的正确性。

第 7 章

位运算程序

逻辑运算和移位指令也称位运算指令,它们都是按位进行的且各位之间相互独立。本章主要介绍逻辑运算指令、移位指令的用法,以及位运算指令在程序设计中的应用,要求熟练掌握指令的功能,了解 ASCII 码和非压缩 BCD 码之间的相互转换关系。

7.1 逻辑运算指令

1. 逻辑非指令

格式:`NOT OPR`

执行操作:对操作数 OPR 按位取反。单操作数指令,操作数不能使用立即数和段寄存器。该指令不影响状态标志位。

2. 逻辑与指令

格式:`AND DST, SRC`

执行操作:将目的操作数 DST 和源操作数 SRC 按位进行与运算,结果存放在目的操作数中。

3. 逻辑或指令

格式:`OR DST, SRC`

执行操作:将目的操作数 DST 和源操作数 SRC 按位进行或运算,结果存放在目的操作数中。

4. 异或指令

格式:`XOR DST, SRC`

执行操作:将目的操作数 DST 和源操作数 SRC 按位进行异或运算,结果存放在目的操作数中。

5. 测试指令

格式:`TEST OPR1, OPR2`

执行操作:将操作数 OPR1 和操作数 OPR2 按位进行与运算,结果不回送给任何操作数,只影响标志位。

6. 关于逻辑运算指令使用的几点说明

(1) 逻辑运算指令,除 NOT 外,均影响标志位,使 CF=0。

(2) 逻辑运算指令用于在处理操作数的某些位。例如:AND 指令用于屏蔽某一特定的位(强制为 0)。OR 指令用于使特定的位置 1。XOR 可以确定两个操作数有哪些位不同,或使操作数的某些位取反。

（3）TEST 指令对两个操作数进行与运算，结果不保存，只影响标志位。

7. 应用举例

```
AND   AL, 0FCH            ;屏蔽 AL 的第 0、1 两位
OR   AL, 20H              ;置 AL 的第 5 位为 1
XOR   AL, 3               ;使 AL 的第 0、1 位变反
AND   AL,AL               ;操作数不变,但影响标志位,使 CF = 0,设置 SF、ZF 等
OR   AL,AL                ;操作数不变,但影响标志位,使 CF = 0,设置 SF、ZF 等
XOR   AL,AL               ;操作数清零,并使 CF = 0
```

7.2 移位指令

移位指令分为两类：非循环移位指令和循环移位指令。指令的一般形式如下：

```
助记符      OPR,CNT
```

其中，OPR 是除立即寻址外的任何操作数，CNT 是移位次数，取 1 或寄存器 CL 内容。

7.2.1 非循环移位指令

1. 逻辑左移（shift logical left）

格式：SHL　OPR,CNT

功能：低位补 0，高位→CF，执行的操作如图 7-1 所示。

2. 算术左移（shift arithmetic left）

格式：SAL　OPR,CNT

功能：低位补 0，高位→CF（与 SHL 完全相同）。

3. 逻辑右移（shift logical right）

格式：SHR　OPR,CNT

功能：高位补 0，低位→CF，执行的操作如图 7-2 所示。

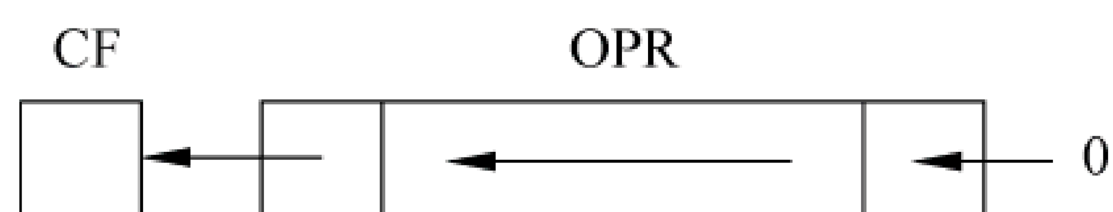


图 7-1　逻辑左移指令

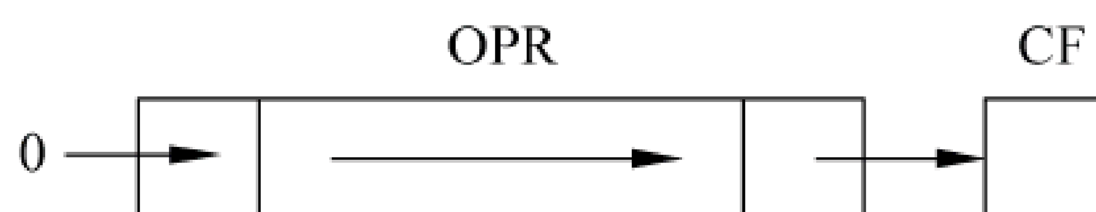


图 7-2　逻辑右移指令

4. 算术右移（shift arithmetic right）

格式：SAR　OPR,CNT

功能：低位→CF，符号位不变，执行的操作如图 7-3 所示。

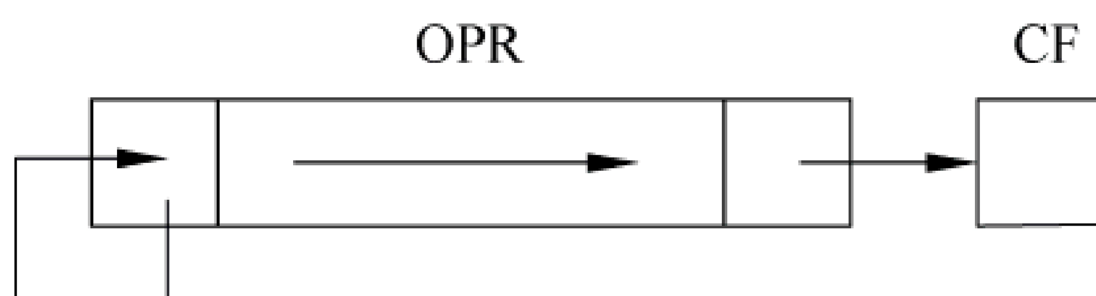


图 7-3　算术右移指令

【例 7-1】　下列两条指令执行后，(AL)＝？

```
MOV   AL,10010110B
SAR   AL,1
```

结果：(AL)＝11001011B。

说明：SAR 右移 1 位，相当于带符号数除以 2；SHR 右移 1 位，相当于无符号数除以 2。

【例 7-2】 下列指令执行后,(AL)=?

```
MOV  CL,4
MOV  AL,05H
SHL  AL,CL
```

结果:(AL)=50H,左移 1 位是无符号数乘以 2,左移 4 位相当于乘以 16。逻辑左移与算术左移指令功能相同。在 DEBUG 系统中,只能使用 SHL 指令,SAL 指令不能被识别。

7.2.2 循环移位指令

1. 循环左移(rotate left)

格式:ROL OPR, CNT

功能:小循环左移,最高位同时送给最低位和 CF,执行的操作如图 7-4 所示。

2. 循环右移(rotate right)

格式:ROR OPR, CNT

功能:小循环右移,最低位同时送给最高位和 CF,执行的操作如图 7-5 所示。

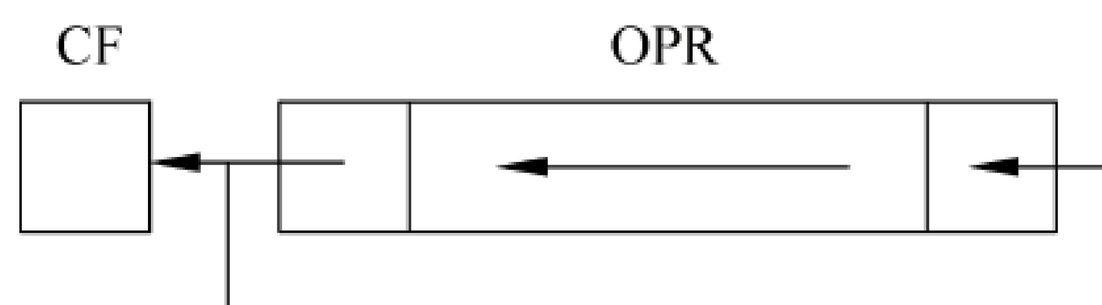


图 7-4 循环左移指令

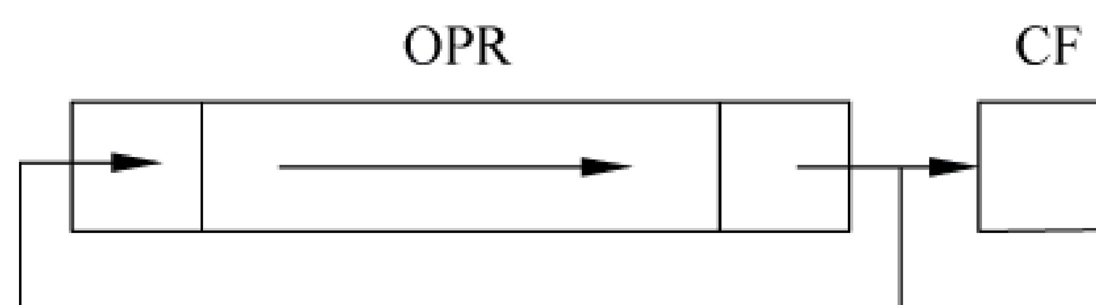


图 7-5 循环右移指令

3. 带进位循环左移(rotate left through carry)

格式:RCL OPR, CNT

功能:大循环左移,执行的操作如图 7-6 所示。

4. 带进位循环右移(rotate right through carry)

格式:RCR OPR, CNT

功能:大循环右移,执行的操作如图 7-7 所示。

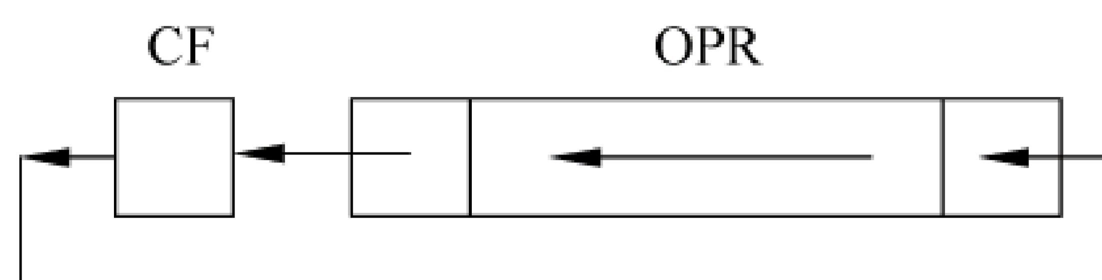


图 7-6 带进位循环左移指令

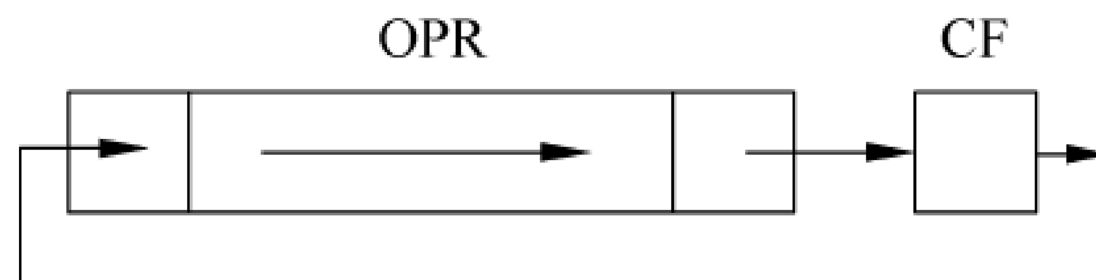


图 7-7 带进位循环右移指令

【例 7-3】 已知 BX=84F0H,分别完成下列问题:

(1) 若(BX)为无符号数,求(BX)/2;

(2) 若(BX)为无符号数,求(BX)×2;

(3) 若(BX)为带符号数,求(BX)/4。

由于逻辑右移 1 位,相当于无符号数除以 2。算术右移 1 位,相当于带符号数除以 2。左移 1 位是无符号数乘以 2,因此可以用移位指令完成上述问题。

(1) 执行“SHR BX, 1”指令,结果 BX = 4278H。DEBUG 下操作结果如下:

```
-R BX
```

```
BX 0000
:84F0
- A
136E:0100 SHR BX,1
136E:0102
- T
AX = 0000   BX = 4278   CX = 0000   DX = 0000   SP = FFEE   BP = 0000   SI = 0000   DI = 0000
DS = 136E   ES = 136E   SS = 136E   CS = 136E   IP = 0102    OV UP EI PL NZ NA PE NC
136E:0102 0000              ADD       [BX + SI],AL                   DS:4278 = 00
-
```

(2) 执行“SAL/SHL　BX，1”指令，结果 BX ＝ 09E0H，CF＝1。DEBUG 下操作结果如下：

```
- R BX
BX 0000
:84F0
- A
136E:0100 SHL BX,1
136E:0102
- T
AX = 0000   BX = 09E0   CX = 0000   DX = 0000   SP = FFEE   BP = 0000   SI = 0000   DI = 0000
DS = 136E   ES = 136E   SS = 136E   CS = 136E   IP = 0102    OV UP EI PL NZ NA PO CY
136E:0102 0000              ADD       [BX + SI],AL                   DS:09E0 = 00
-
```

(3) 执行“MOV　CL,2”和“SAR　BX，CL”，结果 BX ＝ E13CH。DEBUG 下操作结果如下：

```
- R BX
BX 0000
:84F0
- A
136E:0100 MOV CL,2
136E:0102 SAR BX,CL
136E:0104
- T
AX = 0000   BX = 84F0   CX = 0002   DX = 0000   SP = FFEE   BP = 0000   SI = 0000   DI = 0000
DS = 136E   ES = 136E   SS = 136E   CS = 136E   IP = 0102    NV UP EI PL NZ NA PO NC
136E:0102 D3FB              SAR       BX,CL
- T
AX = 0000   BX = E13C   CX = 0002   DX = 0000   SP = FFEE   BP = 0000   SI = 0000   DI = 0000
DS = 136E   ES = 136E   SS = 136E   CS = 136E   IP = 0104    NV UP EI NG NZ NA PE NC
136E:0104 0000              ADD       [BX + SI],AL                   DS:E13C = 00
-
```

【例 7-4】 把 DX:AX 组成的 32 位数乘以 16 运算，写出指令序列。

实际上，乘以 16 就是左移 4 位，编写指令序列如下：

```
MOV   CL,4
SHL   DX,CL
```

```
MOV   BH,AH
SHL   AX,CL
SHR   BH,CL
OR    DL,BH
```

上机实验时，不妨设 DX＝1234H，AX＝5678H。这样组成的 32 位数是 12345678H，乘以 16 后，变成 23456780H。在 DEBUG 下，先用 R 命令设置好寄存器的初值，然后逐条运行每条指令，观察指令的执行结果。

```
-A
136E:0100 MOV CL,4
136E:0102 SHL DX,CL
136E:0104 MOV BH,AH
136E:0106 SHL AX,CL
136E:0108 SHR BH,CL
136E:010A OR DL,BH
136E:010C
-T
AX=5678  BX=0000  CX=0004  DX=1234  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0102   NV UP EI PL NZ NA PO NC
136E:0102 D3E2          SHL     DX,CL
-T
AX=5678  BX=0000  CX=0004  DX=2340  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0104   NV UP EI PL NZ NA PO CY
136E:0104 88E7          MOV     BH,AH
-T
AX=5678  BX=5600  CX=0004  DX=2340  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0106   NV UP EI PL NZ NA PO CY
136E:0106 D3E0          SHL     AX,CL
-T
AX=6780  BX=5600  CX=0004  DX=2340  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0108   OV UP EI PL NZ NA PO CY
136E:0108 D2EF          SHR     BH,CL
-T
AX=6780  BX=0500  CX=0004  DX=2340  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=010A   NV UP EI PL NZ NA PE NC
136E:010A 08FA          OR     DL,BH
-T
AX=6780  BX=0500  CX=0004  DX=2345  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=010C   NV UP EI PL NZ NA PO NC
136E:010C 0000          ADD     [BX+SI],AL                    DS:0500=00
-
```

7.3 位运算指令应用

【例 7-5】 ASCII 码转换成非压缩 BCD 码。假设在 BUF_ASC 单元有一个十进制数字(0～9)的字符，要求将它转换成非压缩 BCD 码的形式存储在 BUF_BCD 单元中。

转换方法：将数字字符的 ASCII 码的高四位清零，即转换为对应的非压缩 BCD 码。反

过来，非压缩 BCD 码加上 30H 即可转换成对应的字符 ASCII 码。

程序如下：

```
DATAS SEGMENT
  BUF_ASC DB '1'
  BUF_BCD DB ?
DATAS ENDS
CODES SEGMENT
    ASSUME CS:CODES,DS:DATAS
START: MOV AX,DATAS
       MOV DS,AX
       MOV AL,BUF_ASC
       AND AL,0FH        ;ASCII 码转换为非压缩 BCD 码
       MOV BUF_BCD,AL
       MOV AH,4CH
       INT 21H
CODES ENDS
      END START
```

在集成实验环境下，通过单步运行，查看运行结果如下：

```
-T
AX=13C5  BX=0000  CX=0021  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13C6  IP=0003   NV UP EI PL NZ NA PO NC
13C6:0003 8ED8           MOV      DS,AX
-T
AX=13C5  BX=0000  CX=0021  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0005   NV UP EI PL NZ NA PO NC
13C6:0005 A00000         MOV      AL,[0000]                  DS:0000=31
-T
AX=1331  BX=0000  CX=0021  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0008   NV UP EI PL NZ NA PO NC
13C6:0008 240F           AND      AL,0F
-T
AX=1301  BX=0000  CX=0021  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=000A   NV UP EI PL NZ NA PO NC
13C6:000A A20100         MOV      [0001],AL                  DS:0001=00
-T
AX=1301  BX=0000  CX=0021  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=000D   NV UP EI PL NZ NA PO NC
13C6:000D B44C           MOV      AH,4C
-D DS:0 L2
13C5:0000  31 01                                          1.
-
```

在数据段(13C5：0000)单元存储的字符'1'的 ASCII 码，转换成了非压缩 BCD 码 01H，存储在(13C5：0001)单元。

7.4 实验内容

【实验目的】 理解和掌握逻辑运算和移位操作指令的功能，了解非压缩 BCD 码和 ASCII 码之间的转换关系，能运用逻辑与移位指令编写程序。

【实验 7-1】 逻辑和移位指令基本功能。

(1) 上机跟踪执行下列程序段每条指令的运行结果。

```
MOV  AX, 0         ;AX = ________,OF = ________,SF = ________,ZF = ________,CF = ________
DEC  AX            ;AX = ________,OF = ________,SF = ________,ZF = ________,CF = ________
ADD  AX, 7FFFH     ;AX = ________,OF = ________,SF = ________,ZF = ________,CF = ________
ADD  AX, 2         ;AX = ________,OF = ________,SF = ________,ZF = ________,CF = ________
NOT  AX            ;AX = ________,OF = ________,SF = ________,ZF = ________,CF = ________
SUB  AX, 1         ;AX = ________,OF = ________,SF = ________,ZF = ________,CF = ________
AND  AX, 58D1H     ;AX = ________,OF = ________,SF = ________,ZF = ________,CF = ________
SAL  AX, 1         ;AX = ________,OF = ________,SF = ________,ZF = ________,CF = ________
SAR  AX, 1         ;AX = ________,OF = ________,SF = ________,ZF = ________,CF = ________
NEG  AX            ;AX = ________,OF = ________,SF = ________,ZF = ________,CF = ________
ROR  AX, 1         ;AX = ________,OF = ________,SF = ________,ZF = ________,CF = ________
```

(2) 若 AL=0FFH,BL=03H,指出下列每条指令执行后的状态标志位的值。

```
① ADD  BL, AL     ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
② INC  BL         ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
③ SUB  BL, AL     ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
④ NEG  BL         ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
⑤ CMP  BL , AL    ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
⑥ MUL  BL         ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
⑦ AND  BL, AL     ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
⑧ IMUL  BL        ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
⑨ OR  BL, AL      ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
⑩ SHL  BL, 1      ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
⑪ XOR  BL, BL     ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
⑫ SAR  AL, 1      ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
⑬ SHR  AL, 1      ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
⑭ XCHG  BL, AL    ;OF = ______,SF = ______,ZF = ______,AF = ______,PF = ______,CF = ______
```

【实验 7-2】 将非压缩 BCD 码转换成 ASCII 码。将 BUF 单元存储一位十进制数字(非压缩 BCD 码)显示在屏幕上。

这实际上是非压缩 BCD 码转换成 ASCII 码的问题,将非压缩 BCD 码得高四位拼上 3,就能转换成对应的 ASCII 码,然后利用 2 号 DOS 功能调用(相关内容参见第 12 章)即可显示输出。程序如下:

```
DATAS SEGMENT
    BUF DB 05H
DATAS ENDS
CODES SEGMENT
     ASSUME CS:CODES,DS:DATAS
START: MOV AX,DATAS
        MOV DS,AX
        MOV DL,BUF
        OR  DL,30H    ;非压缩 BCD 码转换成 ASCII 码
```

```
        MOV AH,2
        INT 21H
        MOV AH,4CH
        INT 21H
CODES ENDS
        END START
```

习　　题

1. 分析下列指令的运行结果并通过上机验证。

```
(1) MOV   AX,00ABH
    MOV   BX,0037H
    MOV   CL,8
    ROL   BX,CL
    ADD   AX,BX       ;AX = ?
(2) MOV   AL, 57H
    XOR   AL,0FH      ;AL = ?   CF = ?
(3) XOR   AX,AX
    MOV   AX,2017H
    MOV   CX,0404H
    ROL   AH,CL
    XCHG  CH,CL
    ROR   AL,CL       ;AX = ?   CF = ?
(4) MOV   CL,3
    MOV   BX,0B7H
    ROL   BX,1
    ROR   BX,CL       ;BX = ?
```

2. 已知 AX=0C2H,BX=0AFH,CX=0A02H,CF=1,逐条运行以下指令序列,写出每条指令执行后有关寄存器和 CF 的内容,然后上机验证。

```
TEST   AX,BX
SAL    AX,1
SHR    BX,CL
ROR    AX,1
ROL    BX,CL
```

3. 分析下面的程序段完成什么功能。

```
MOV   CL,4
SHL   DX,CL
MOV   BL,AH
SHL   AX,CL
SHR   BL,CL
OR    DL,BL
```

第 8 章

串操作

串操作指令处理的对象是一片连续的字节或字数据存储单元，每次基本操作后能够自动修改地址指针。串操作指令与重复前缀配合，可以实现循环操作的功能，是实际应用中非常有用的一组指令。本章主要介绍串操作指令及其程序设计，要求掌握串操作指令的具体格式、功能和用法，能够编写串操作处理程序。

8.1 串操作指令

串操作指令是对字节或字组成的数据且在内存中连续存储的字节串或字串进行操作，每次基本操作仅处理一个元素（字节或字）。串操作指令包括：MOVS 串传送指令、LODS 串装入指令、STOS 串存储指令、CMPS 串比较指令和 SCAS 串扫描指令。

串操作指令的共性特点：

（1）用 SI 寄存器存放源操作数（SRC）的偏移地址，用 DI 寄存器存放目的操作数（DST）的偏移地址。

（2）源串隐含在数据段（DS），目标串隐含在附件段（ES）。

（3）每一次基本操作后，串指令自动修改地址指针。

（4）串指令可加重复前缀 REP，重复次数由 CX 指定。

8.1.1 MOVS、LODS、STOS 指令

1. 串传送指令 MOVS（move string）

格式

```
MOVS   DST,SRC
MOVSB               ;字节串传送
MOVSW               ;字串传送
```

该指令是将（SI）指定的存储单元的内容传送到（DI）指定的存储单元中，执行的操作如下：

```
(ES: DI)←(DS: SI)
SI±1,DI±1           ;字节操作
SI±2,DI±2           ;字操作
```

其中，当方向标志 DF=0 时，用“+”号；当 DF=1 时，用“-”号。串传送指令不影响标志位。如果要按增址方向操作，要使用 CLD 指令，设置 DF=0；如果要按减址方向操作，要使用 STD 指令，设置 DF=1。

2. 串装入指令 LODS(load from string)

格式:

```
LODS   SRC
LODSB                 ;从字节串取
LODSW                 ;从字串取
```

该指令是将(SI)指定的存储单元的内容传送到 AL 或 AX 中,执行的操作如下:

```
AL←(SI),SI±1          ;字节操作
AX←(SI),SI±2          ;字操作
```

其中,当方向标志 DF=0 时,用"+"号;当 DF=1 时,用"-"号。LODS 指令不影响状态标志。

3. 串存储指令 STOS(store into string)

格式:

```
STOS   DST
STOSB                 ;存入字节串
STOSW                 ;存入字串
```

该指令是把 AL 或 AX 的内容存入到(DI)指定的存储单元中,执行的操作如下:

```
(DI)←AL,DI±1          ;字节操作
(DI)←AX,DI±2          ;字操作
```

其中,当方向标志 DF=0 时,用"+"号;当 DF=1 时,用"-"号。STOS 指令不影响状态标志。

4. 重复串操作前缀 REP(repeat)

REP 可与 MOVS、LODS、STOS 指令配合,使 REP 前缀后面的串操作指令重复执行,执行的次数要事先存储在 CX 中。执行的操作如下:

(1) 若 CX=0,则退出 REP;否则,继续执行;

(2) CX←CX-1;

(3) 执行 REP 前缀后面的串操作指令 1 次;

(4) 重复执行(1)~(3)的操作。

重复串操作前缀 REP 与串操作指令配合使用,不仅可以简化程序,而且可以提高运行速度。如果不使用 REP,需要使用循环结构来完成相同的操作。

8.1.2 CMPS 和 SCAS 指令

1. 串比较指令 CMPS(compare string)

格式:

```
CMPS   SRC,DST
CMPSB                 ;字节串比较
CMPSW                 ;字串比较
```

该指令是将两个串中相应的元素逐个进行比较,执行源操作数 SRC 减去目的操作数

DST，但不保存结果，只反映在状态标志位上。执行的操作：

```
(SI)-(DI)
SI±1,DI±1          ;字节操作
SI±2,DI±2          ;字操作
```

其中，当方向标志 DF=0 时，用“+”号；当 DF=1 时，用“-”号。

2. 串扫描指令 SCAS(scan string)

格式：

```
SCAS   DST
SCASB              ;字节串扫描
SCASW              ;字串扫描
```

该指令是将累加器 AL/AX 的内容与串中的元素逐个进行比较，比较结果不保存，只反映在状态标志位上。执行的操作：

```
AL-(DI),DI±1       ;字节操作
AX-(DI),DI±2       ;字操作
```

其中，当方向标志 DF=0 时，用“+”号；当 DF=1 时，用“-”号。待搜索的关键字必须放在 AL 或 AX 中。

3. 当相等/为零时重复操作前缀 REPE/REPZ(repeat while equal/zero)

格式：REPE /REPZ

重复操作的条件：CX≠0 且 ZF=1。

REPE /REPZ 可与 CMPS、SCAS 指令配合使用，当 CX=0 或 ZF=0 时，退出串比较或串扫描。否则，继续重复操作。

4. 当不相等/不为零时重复操作前缀 REPE/REPZ(repeat while not equal/not zero)

格式：REPNE /REPNZ

重复操作的条件：CX≠0 且 ZF=0。

REPNE /REPNZ 可与 CMPS、SCAS 指令配合使用，当 CX=0 或 ZF=1 时，退出串比较或串扫描。否则，继续重复操作。

8.2 串操作程序

【例 8-1】 把数据段中首地址为 S1 的 30 个字节数据传送到附加段中的首地址为 S2 的一片连续存储单元中。

在本例中，可以用串操作指令实现，也可以用一般的 MOV 指令实现。用串操作指令，本例给出了四种不同的方法，目的是全面理解各种串操作指令的用法。如果不用串操作指令，只能用一般的 MOV 指令。由于方法 3 和方法 4 都涉及了循环结构，如果不能理解程序结构，可以参阅第 10 章循环控制语句的有关内容。

```
;方法 1: 使用 REP MOVSB 指令
DATA1 SEGMENT
   S1   DB 'ABCDEFGHIJKLMNOPQRSTUV'
```

```
        DB 'WXYZ0123'
DATA1 ENDS
DATA2 SEGMENT
   S2  DB 30 DUP(?)
DATA2 ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA1,ES:DATA2
START: MOV AX,DATA1
       MOV DS,AX
       MOV AX,DATA2
       MOV ES,AX
       MOV SI,OFFSET S1
       MOV DI,OFFSET S2
       MOV CX,30
       CLD                        ;设置 DF = 0
       REP MOVSB
       MOV AH,4CH
       INT 21H
CODE ENDS
       END START
;方法 2: 使用 REP MOVS S2,S1 指令
DATA1 SEGMENT
   S1   DB 'ABCDEFGHIJKLMNOPQRSTUV'
        DB 'WXYZ0123'
DATA1 ENDS
DATA2 SEGMENT
   S2   DB 30 DUP(?)
DATA2 ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA1,ES:DATA2
START: MOV AX,DATA1
       MOV DS,AX
       MOV AX,DATA2
       MOV ES,AX
       MOV CX,30
       CLD
       REP MOVS S2,S1
       MOV AH,4CH
       INT 21H
CODE ENDS
       END START

;方法 3: 使用 LODSB 和 STOSB 指令
DATA1 SEGMENT
   S1  DB 'ABCDEFGHIJKLMNOPQRSTUV'
       DB 'WXYZ0123'
DATA1 ENDS
DATA2 SEGMENT
   S2  DB 30 DUP(?)
DATA2 ENDS
CODE SEGMENT
```

```
     ASSUME CS:CODE,DS:DATA1,ES:DATA2
START:  MOV AX,DATA1
        MOV DS,AX
        MOV AX,DATA2
        MOV ES,AX
        LEA SI,S1
        LEA DI,S2
        MOV CX,30
        CLD
NEXT:   LODSB                       ;串装入 AL
        STOSB                       ;存入字节串
        LOOP NEXT
        MOV AH,4CH
        INT 21H
CODE ENDS
        END START
```

```
;方法 4: 用一般的 MOV 指令
DATA1 SEGMENT
   S1    DB 'ABCDEFGHIJKLMNOPQRSTUV'
         DB 'WXYZ0123'
DATA1 ENDS
DATA2 SEGMENT
   S2    DB 30 DUP(?)
DATA2 ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:DATA1,ES:DATA2
START:  MOV AX,DATA1
        MOV DS,AX
        MOV AX,DATA2
        MOV ES,AX
        LEA SI,S1
        LEA DI,S2
        MOV CX,30
NEXT:   MOV AL,[SI]                 ;从串取一个元素送给 AL
        MOV ES:[DI],AL              ;存入
        INC SI
        INC DI                      ;调整地址指针
        LOOP NEXT
        MOV AH,4CH
        INT 21H
CODE ENDS
        END START
```

8.3 实验内容

【实验目的】 深入理解串操作指令和重复操作前缀指令的应用，能够运用串操作指令编写应用程序，并掌握串操作的上机调试方法。

【实验 8-1】 把数据段中首地址为 S1 的 30 个字节数据传送到同一数据段中的首地址为 S2 的一片连续存储单元中。按照如下要求分别完成编程，并通过调试程序查看串操作指令执行前后 S1、S2 存储区内容的变化。

(1) 使用串操作指令，按增址方向传送字符串。

(2) 使用串操作指令，按减址方向传送字符串。

(3) 若 S1 末端和 S2 首端有部分存储区重叠，不妨设 S2 的起始位置在 S1 的第 21 个字符存储单元处，应如何实现？

(4) 若 S1 首端和 S2 末端有部分存储区重叠，不妨设 S1 的起始位置在 S2 的第 21 个字符存储单元处，又应如何实现？

解析：

(1) 按增址方向传送字符串，可以仿照例 8-1 的方法实现，只不过这里 DS 和 ES 共用同一个段。

程序如下：

```
DATA1 SEGMENT
   S1    DB 'ABCDEFGHIJKLMNOPQRSTUV'
         DB 'WXYZ0123'
   S2    DB 30 DUP(?)
DATA1 ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:DATA1
START:  MOV AX,DATA1
        MOV DS,AX
        MOV ES,AX
        MOV SI,OFFSET S1
        MOV DI,OFFSET S2
        MOV CX,30
        CLD                          ;设置 DF = 0
        REP MOVSB
        MOV AH,4CH
        INT 21H
CODE ENDS
        END START
```

在 DEBUG 中单步运行时，遇到重复串操作指令时，如果用 T 命令，它会多次重复执行串操作。这时最好使用 P 命令，一步执行完所有的重复串操作。串指令执行后查看存储区内容的变化情况，运行结果如下：

```
 -P
AX = 13C5   BX = 0000   CX = 001E   DX = 0000   SP = 0000   BP = 0000   SI = 0000   DI = 001E
DS = 13C5   ES = 13C5   SS = 13C5   CS = 13C9   IP = 0011    NV UP EI PL NZ NA PO NC
13C9:0011 F3                  REPZ
13C9:0012 A4                  MOVSB
 -P
AX = 13C5   BX = 0000   CX = 0000   DX = 0000   SP = 0000   BP = 0000   SI = 001E   DI = 003C
DS = 13C5   ES = 13C5   SS = 13C5   CS = 13C9   IP = 0013    NV UP EI PL NZ NA PO NC
13C9:0013 B44C                MOV        AH,4C
```

```
-D DS:0
13C5:0000  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50   ABCDEFGHIJKLMNOP
13C5:0010  51 52 53 54 55 56 57 58-59 5A 30 31 32 33 41 42   QRSTUVWXYZ0123AB
13C5:0020  43 44 45 46 47 48 49 4A-4B 4C 4D 4E 4F 50 51 52   CDEFGHIJKLMNOPQR
13C5:0030  53 54 55 56 57 58 59 5A-30 31 32 33 00 00 00 00   STUVWXYZ0123....
13C5:0040  B8 C5 13 8E D8 8E C0 BE-00 00 BF 1E 00 B9 1E 00   ................
13C5:0050  FC F3 A4 B4 4C CD 21 00-00 00 00 00 00 00 00 00   ....L.!.........
13C5:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

(2) 如果按减址方向传送，指向 S1、S2 的地址指针，开始就要把地址指针定位在字符串存储的最后一个单元上。

程序如下：

```
DATA SEGMENT
   S1    DB 'ABCDEFGHIJKLMNOPQRSTUV'
         DB 'WXYZ0123'
   S2    DB 30 DUP(?)
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV ES,AX
        MOV SI,OFFSET S1+29
        MOV DI,OFFSET S2+29
        MOV CX,30
        STD                          ;设置 DF=1
        REP MOVSB
        MOV AH,4CH
        INT 21H
CODE ENDS
        END START
```

(3) 数据段定义时，S1 的前 20 个字符不发生重叠现象，后 10 个字符与 S2 存储区前 10 个单元有重叠，即两片存储区有 10 个存储单元发生重叠。可以仿照按减址方向传送字符串的方法实现。

程序如下：

```
DATA SEGMENT
   S1    DB 'ABCDEFGHIJKLMNOPQRST'
   S2    DB 'UVWXYZ0123'
          DB 20 DUP(?)
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV ES,AX
```

```
        MOV SI,OFFSET S1 + 29
        MOV DI,OFFSET S2 + 29
        MOV CX,30
        STD                          ;设置 DF = 1
        REP MOVSB
        MOV AH,4CH
        INT 21H
CODE ENDS
        END START
```

串指令执行后,查看存储区内容的变化情况,运行结果如下:

```
-P
AX = 13C5  BX = 0000  CX = 001E  DX = 0000  SP = 0000  BP = 0000  SI = 001D  DI = 0031
DS = 13C5  ES = 13C5  SS = 13C5  CS = 13C9  IP = 0011   NV DN EI PL NZ NA PO NC
13C9:0011 F3              REPZ
13C9:0012 A4              MOVSB
-P
AX = 13C5  BX = 0000  CX = 0000  DX = 0000  SP = 0000  BP = 0000  SI = FFFF  DI = 0013
DS = 13C5  ES = 13C5  SS = 13C5  CS = 13C9  IP = 0013   NV DN EI PL NZ NA PO NC
13C9:0013 B44C            MOV      AH,4C
-D DS:0
13C5:0000  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50   ABCDEFGHIJKLMNOP
13C5:0010  51 52 53 54 41 42 43 44-45 46 47 48 49 4A 4B 4C   QRSTABCDEFGHIJKL
13C5:0020  4D 4E 4F 50 51 52 53 54-55 56 57 58 59 5A 30 31   MNOPQRSTUVWXYZ01
13C5:0030  32 33 00 00 00 00 00 00-00 00 00 00 00 00 00 00   23..............
13C5:0040  B8 C5 13 8E D8 8E C0 BE-1D 00 BF 31 00 B9 1E 00   ...........1....
13C5:0050  FD F3 A4 B4 4C CD 21 00-00 00 00 00 00 00 00 00   ....L.!.........
13C5:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

(4) 数据段定义时,S2 的前 20 个存储单元不发生重叠现象,后 10 个单元与 S1 存储区的前 10 个字符存储单元有重叠,这种情形可以按增址方向传送字符串的方法实现。

程序如下:

```
DATA SEGMENT
   S2   DB 20 DUP(?)
   S1   DB 'ABCDEFGHIJKLMNOPQRST'
        DB 'UVWXYZ0123'
DATA ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV ES,AX
        MOV SI,OFFSET S1
        MOV DI,OFFSET S2
        MOV CX,30
        CLD                          ;设置 DF = 0
        REP MOVSB
```

```
        MOV AH,4CH
        INT 21H
CODE ENDS
        END START
```

串指令执行后,查看存储区内容的变化情况,运行结果如下:

```
-P
AX=13C5  BX=0000  CX=001E  DX=0000  SP=0000  BP=0000  SI=0014  DI=0000
DS=13C5  ES=13C5  SS=13C5  CS=13C9  IP=0011   NV UP EI PL NZ NA PO NC
13C9:0011 F3            REPZ
13C9:0012 A4            MOVSB
-P
AX=13C5  BX=0000  CX=0000  DX=0000  SP=0000  BP=0000  SI=0032  DI=001E
DS=13C5  ES=13C5  SS=13C5  CS=13C9  IP=0013   NV UP EI PL NZ NA PO NC
13C9:0013 B44C          MOV     AH,4C
-D DS:0
13C5:0000  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50  ABCDEFGHIJKLMNOP
13C5:0010  51 52 53 54 55 56 57 58-59 5A 30 31 32 33 4B 4C  QRSTUVWXYZ0123KL
13C5:0020  4D 4E 4F 50 51 52 53 54-55 56 57 58 59 5A 30 31  MNOPQRSTUVWXYZ01
13C5:0030  32 33 00 00 00 00 00 00-00 00 00 00 00 00 00 00  23..............
13C5:0040  B8 C5 13 8E D8 8E C0 BE-14 00 BF 00 00 B9 1E 00  ................
13C5:0050  FC F3 A4 B4 4C CD 21 00-00 00 00 00 00 00 00 00  ....L.!.........
13C5:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
-
```

习　　题

1. 现有程序段如下:

```
CLD
LEA SI, BUFFER1                ;BUFFER1 为字节变量
LEA DI, BUFFER2                ;BUFFER2 为字节变量
MOV CX, 100
REP MOVSB
```

该程序段完成的功能是什么?

2. 现有程序段如下:

```
CLD
MOV DI, OFFSET BUFFER
MOV AL, 35H
MOV CX, 100
REPNZ SCASB
```

该程序段完成的功能是什么?

3. 现有程序段如下:

```
MOV AX, 2000H
```

```
MOV ES, AX
LEA DI, BUFFER
XOR AL, AL
MOV CX, 100
REP STOSB
```

该程序段完成的功能是什么？

第9章

分支程序设计

结构化程序的基本结构是顺序结构、分支结构和循环结构，前面介绍的程序基本上都是顺序结构程序，本章主要介绍控制转移指令及分支结构程序，要求重点掌握分支程序结构、条件转移指令的功能，掌握无符号数和有符号数比较的不同判定方法。通过学习，掌握简单分支程序设计和多分支程序设计的方法。

9.1 控制转移指令

汇编指令的执行顺序由 CS 和 IP 的内容确定，即(CS) * 10H＋(IP)，指向将从中取出下一条指令的存储单元。一般情况下，CS 值固定，IP 自动加 1，即指令顺序执行。如果要改变指令的执行顺序，就要改变 CS 或 IP 寄存器的内容。

控制转移指令包括：无条件转移、条件转移、循环控制、中断等。本章主要讨论与分支程序相关的无条件转移指令和条件转移指令，其他内容放在后续章节中介绍。

9.1.1 无条件转移指令

1. 一般指令格式

格式：JMP　目的操作数

功能：无条件地转移到指定的目标地址。

转移类型有两种：

(1) 段内转移：只改变 IP 的内容。

(2) 段间转移：同时改变 CS 和 IP 的内容。

2. 无条件转移指令具体格式

1) 段内直接短转移

格式：JMP　SHORT　标号名

执行操作：(IP) ← (IP) ＋ 8 位位移量。

转移范围：－128～＋127。

2) 段内直接近转移

格式：JMP　NEAR PTR　标号名

执行操作：(IP) ← (IP) ＋ 16 位位移量。

转移范围：－32K～＋32K。

3) 段内间接转移

格式：JMP　WORD PTR　OPR

执行操作：(IP) ← (EA)，有效地址 EA 由 OPR 的寻址方式决定。

例如：

```
JMP   BX                      ;IP←BX
JMP   WORD   PTR [BX]
```

4) 段间直接远转移

格式：JMP　FAR PTR　标号

执行操作：(IP) ←标号所在段中的偏移地址；(CS) ←标号所在段的段地址。

9.1.2　条件转移指令

这类指令是以标志位的状态作为转移条件，若满足条件，则程序转移到目标地址；若不满足，则顺序执行下一条指令。

格式：助记符　　标号名

标号属性：SHORT。

转移范围：－128～＋127。

这类指令不影响标志位，而且只能使用段内直接寻址的 8 位位移量。下面对条件转移指令进行分类讨论，指令中 OPR 代表操作数，通常以标号的形式出现，汇编后形成 8 位补码表示的位移量。

1. 根据单个条件标志的设置情况转移

格式		测试条件
JZ(JE)	OPR	ZF = 1
JNZ(JNE)	OPR	ZF = 0
JS	OPR	SF = 1
JNS	OPR	SF = 0
JO	OPR	OF = 1
JNO	OPR	OF = 0
JP	OPR	PF = 1
JNP	OPR	PF = 0
JC	OPR	CF = 1
JNC	OPR	CF = 0

2. 比较两个无符号数，并根据比较结果转移

	格式		测试条件
＜	JB (JNAE,JC)	OPR	CF = 1
≥	JNB (JAE,JNC)	OPR	CF = 0
≤	JBE (JNA)	OPR	CF∨ZF = 1
＞	JNBE (JA)	OPR	CF∨ZF = 0

3. 比较两个带符号数，并根据比较结果转移

	格式		测试条件
＜	JL (JNGE)	OPR	SF∀OF = 1

≥　JNL (JGE)　OPR　SF∀OF = 0

≤　JLE (JNG)　OPR　(SF∀OF)∨ZF = 1

>　JNLE (JG)　OPR　(SF∀OF)∨ZF = 0

4. 测试 CX 的值为 0 则转移

格式　　　　测试条件

JCXZ　OPR　(CX)=0

9.2 分支结构程序

9.2.1 分支结构的概念

分支结构是根据测试条件的判定结果，从两个或多个分支中选择一个分支的程序段来执行。分支结构分为双分支和多分支两种结构，其共同特点是程序执行方向是向前的，在某种确定的条件下，只能执行其中的一个分支程序段。

双分支结构是根据测试条件是否满足，分别处理不同的分支程序段或者绕过某段程序。图 9-1(a)形式也称单分支结构，图 9-1(b)是典型的双分支结构。

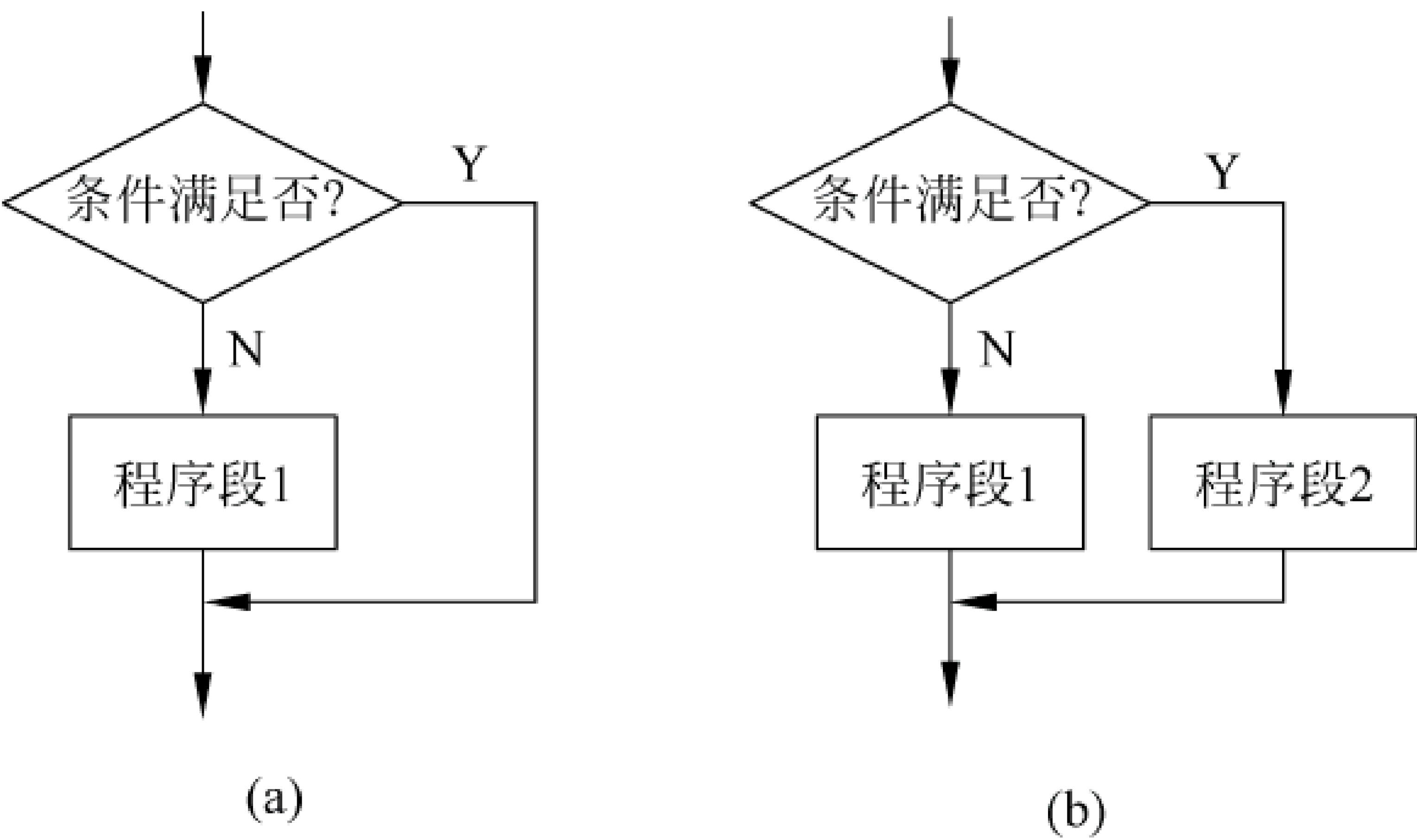


图 9-1　双分支结构框图

多分支结构适用于多种条件的情况，每个条件对应一个程序段，哪个条件满足就转到对应的程序段执行，从而达到根据不同的条件实现多路分支的程序转移，其流程图如图 9-2 所示。

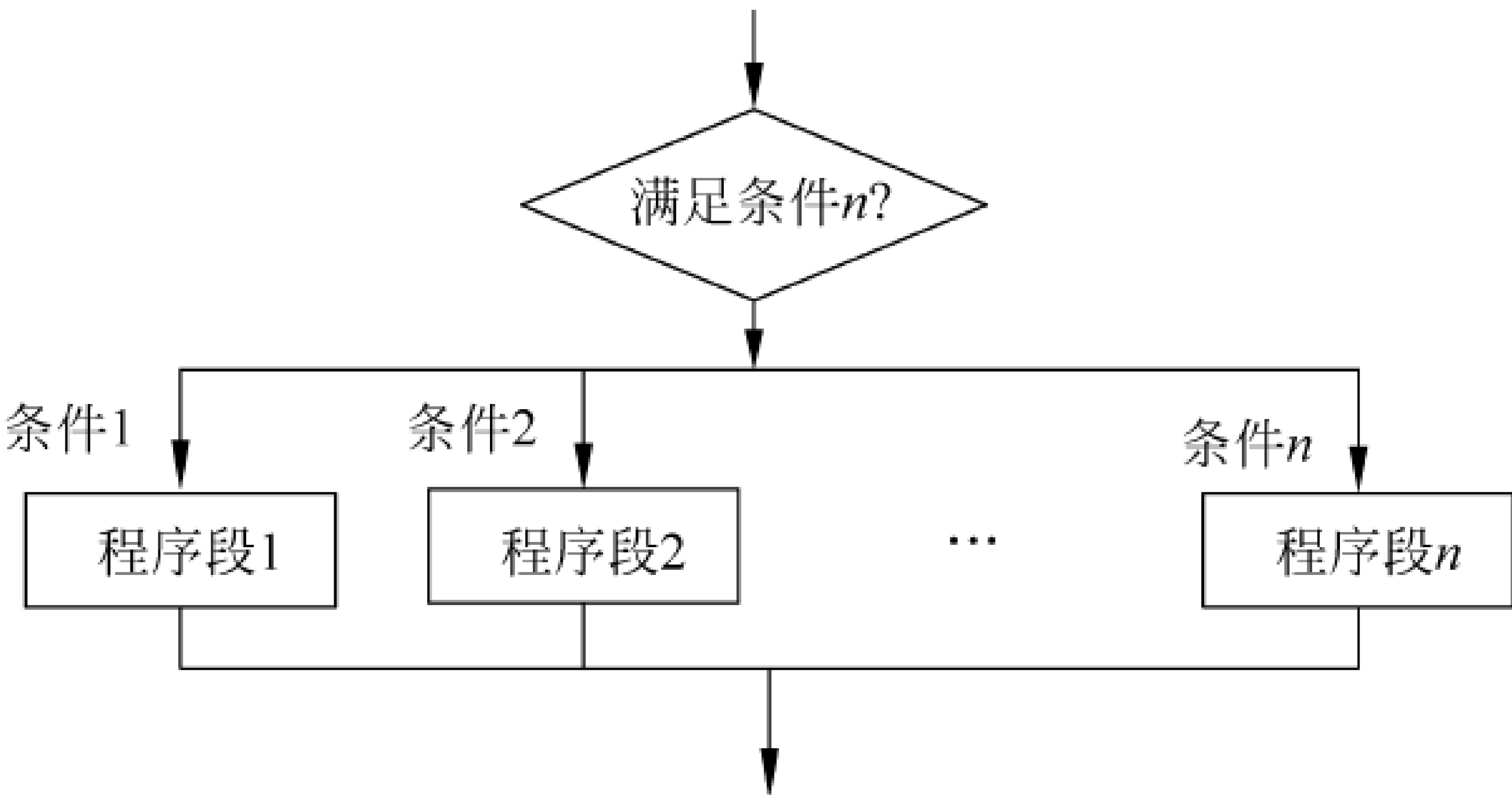


图 9-2　多分支结构框图

9.2.2 双分支程序设计

【例 9-1】 比较两个带符号字节数的大小,两个数中的较小者存入 MIN 字节单元。

比较两个数大小,可以先把第一个数送给 AL,然后用 CMP 指令比较 AL 的内容和第二个数的大小,根据标志位的状态确定分支结构的走向。程序设计流程如图 9-3 所示。

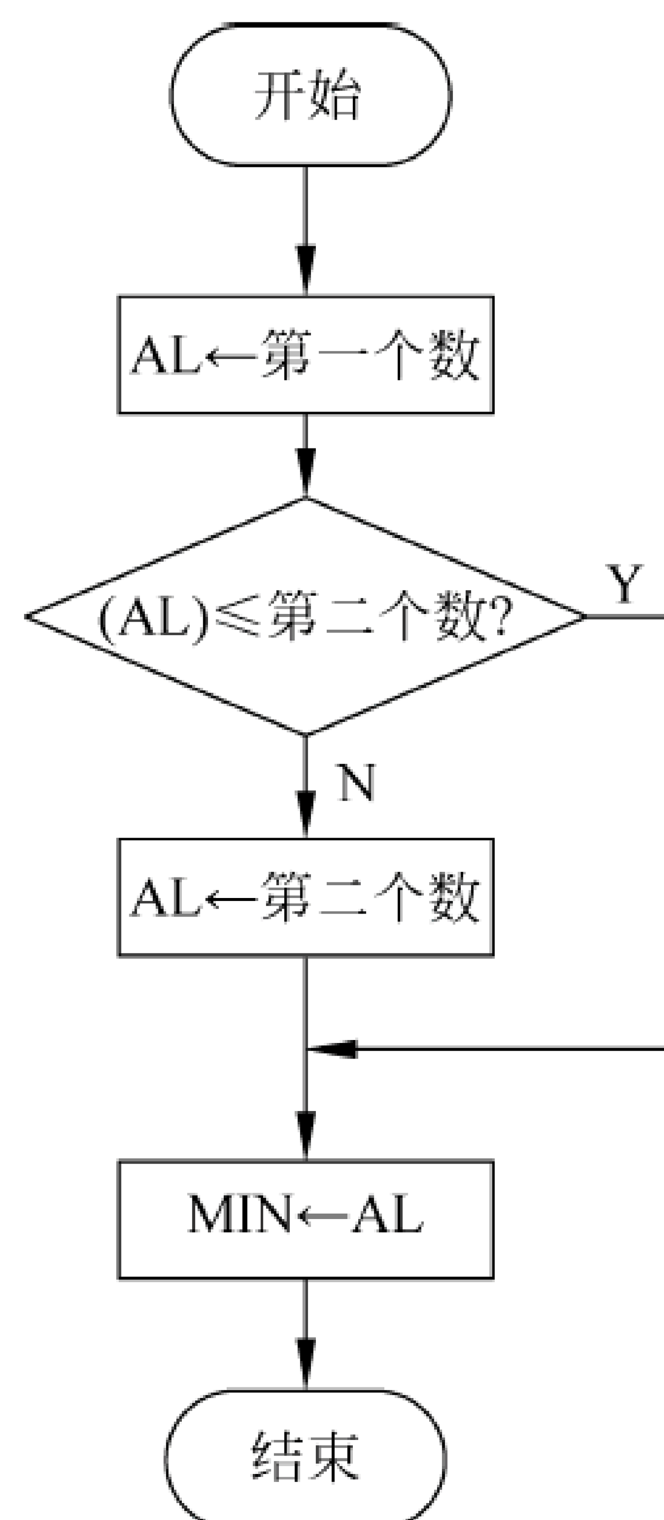


图 9-3 求较小数的程序设计流程图

程序如下:

```
DATA   SEGMENT
    DA1   DB 81H,72H       ;任意给两个带符号数
    MIN   DB ?
DATA   ENDS
CODE   SEGMENT
     ASSUME CS:CODE,DS:DATA
START: MOV     AX,DATA
       MOV     DS,AX
       MOV     AL,DA1
       CMP     AL,DA1 + 1 ;第一个数与第二个数比较
       JLE     L1         ;若小于或等于,则转到 L1,第一个数是较小数
       MOV     AL,DA1 + 1 ;若大于,则第二个数是较小数
L1:    MOV     MIN,AL
       MOV     AH,4CH
       INT     21H
 CODE  ENDS
       END     START
```

【例 9-2】 实现如下的符号函数,其中 $-128<x<127$,x 和 y 均为字节变量,并分别存储在 VAL_X 和 VAL_Y 单元中。

$$y=\begin{cases}1, & x>0\\ 0, & x=0\\ -1, & x<0\end{cases}$$

这是一个具有三个分支的分段函数,可以采用双分支的嵌套结构实现。复杂的分支程序设计,一般先画一个流程图,这样思路清晰,易于编程。本例题的符号函数程序设计流程图如图 9-4 所示。

程序如下:

```
DSEG   SEGMENT
  VAL_X  DB   20H        ;任意给一个数
  VAL_Y  DB   ?
DSEG   ENDS
CSEG   SEGMENT
    ASSUME  CS:CSEG, DS:DSEG
START: MOV  AX, DSEG
       MOV  DS,AX
       MOV  AL,VAL_X
       CMP  AL, 0
```

```
        JGE   BIGGER
        MOV   AL, -1
        JMP   SAVE
BIGGER:JE     EQUL
        MOV   AL, 1
        JMP   SAVE
EQUL:   MOV   AL, 0
SAVE:   MOV   VAL_Y, AL
        MOV   AH,4CH
        INT   21H
CSEG    ENDS
    END START
```

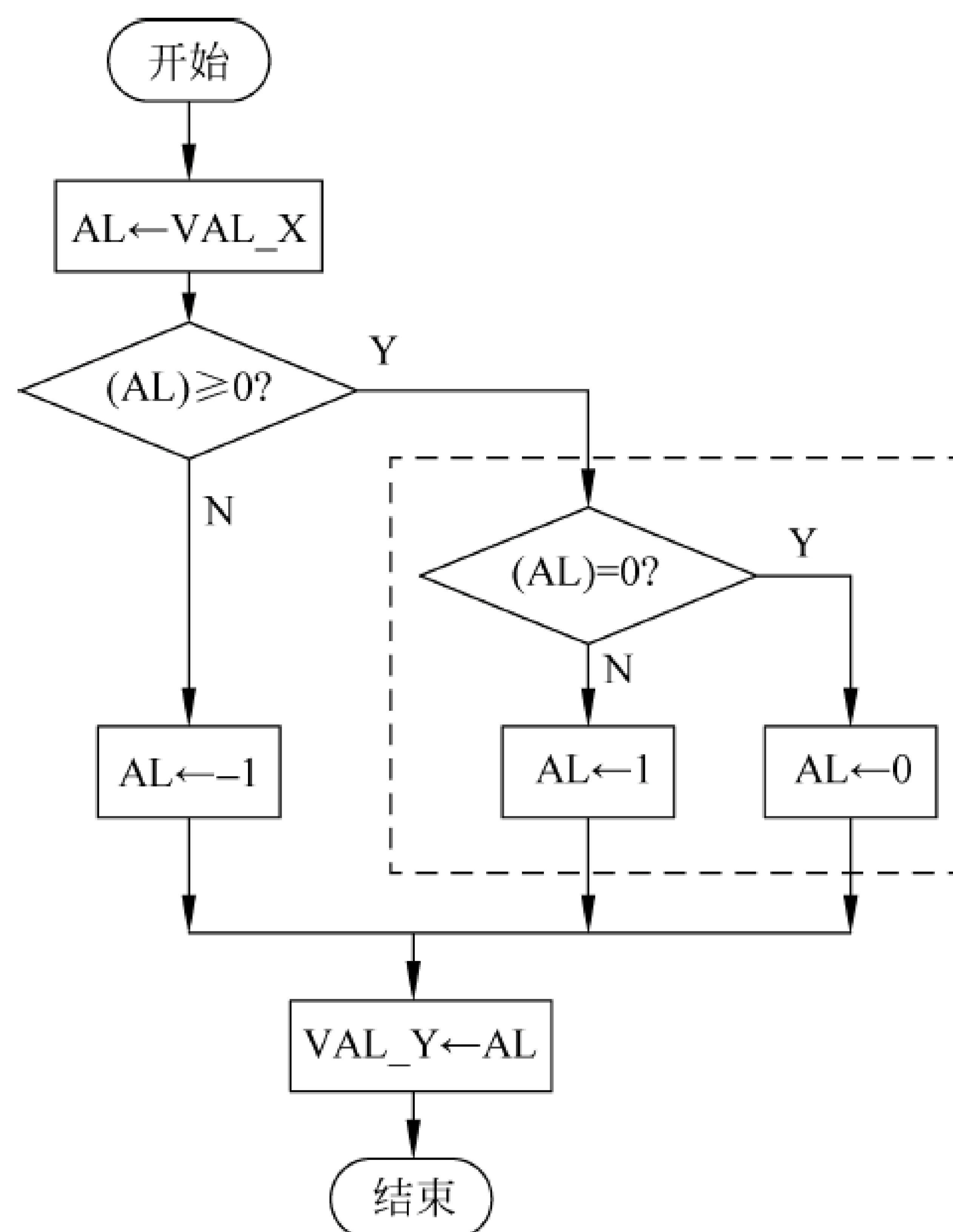


图 9-4　符号函数的程序设计流程图

9.2.3　多分支程序设计

多分支程序设计的关键,是如何按条件对分支进行判断,从而根据不同的条件转移到不同的程序段。在汇编语言中有转移表法、地址表法和逻辑分解法等,实现多分支程序设计。

转移表法是把转移到各分支的转移指令依次存放在代码段的一张表格中,各分支转移指令在表中的位置,即距离表格首地址的位移作为条件。当进行多分支条件判断时,把当前的条件位移量处理后加上转移表的首地址,转移到转移指令表的相应位置,执行转移表中的无条件转移指令,进入相应的程序段。

地址表法是在数据段中把各分支程序段的入口地址建立一个表,取各分支段的编号作为各分支入口地址的表地址的位移量,形成相应程序段的入口地址,从而转移到相应分支的程序段。

逻辑分解法是针对多分支结构中的相容性条件的情况,按照事先约定的优先级,依次查

询,转移到相应的分支程序段。这种方法可以理解成由一串双分支结构嵌套组成。

【例 9-3】 根据从键盘上输入的参数值 0～9,分别转向标号为 L0～L9 的程序段。假设 L0～L9 的程序段分别实现显示英文字母 A～J。

(1) 地址表法:利用 1 号 DOS 功能调用,实现从键盘输入一个参数 0～9,AL 寄存器接收下来是数字字符的 ASCII 码,需要转换成对应的二进制数作为地址表的位移量,形成地址表的偏移地址在 BX 中。最后,利用段内间接转移指令,转移到对应的标号处。显示输出时,采用的是 2 号 DOS 功能调用。关于系统功能调用的详细知识,参见第 12 章。

程序如下:

```
DATA   SEGMENT
TABLE     DW  L0                     ;地址表
          DW  L1
          DW  L2
          DW  L3
          DW  L4
          DW  L5
          DW  L6
          DW  L7
          DW  L8
          DW  L9
DATA   ENDS
CODE   SEGMENT
    ASSUME  CS:CODE, DS:DATA
START: MOV AX, DATA
       MOV DS, AX
       MOV AH, 01H                   ;1 号功能调用,键盘输入
       INT 21H
       CMP AL, 30H
       JB EXIT
       CMP AL, 39H
       JA EXIT
       AND AL, 0FH
       MOV AH,0
       ADD AX,AX                     ;形成位移量
       MOV BX, OFFSET TABLE          ;取地址表首地址
       ADD BX,AX                     ;形成地址表的偏移地址
       JMP [BX]
L0:    MOV DL, 'A'
       JMP DISP
L1:    MOV DL, 'B'
       JMP DISP
L2:    MOV DL, 'C'
       JMP DISP
L3:    MOV DL, 'D'
       JMP DISP
L4:    MOV DL, 'E'
       JMP DISP
L5:    MOV DL, 'F'
       JMP DISP
```

```
L6:     MOV DL, 'G'
        JMP DISP
L7:     MOV DL, 'H'
        JMP DISP
L8:     MOV DL, 'I'
        JMP DISP
L9:     MOV DL, 'J'
        JMP DISP
DISP:   MOV AH,02H                  ;2 号功能调用,显示输出
        INT 21H
EXIT:   MOV AH,4CH
        INT 21H
CODE      ENDS
        END   START
```

(2)转移表法。

```
CODE   SEGMENT
    ASSUME   CS:CODE
START: MOV AH, 01H
        INT 21H                   ;1 号功能调用,键盘输入
        CMP AL, 30H
        JB EXIT
        CMP AL, 39H
        JA EXIT
        AND AL, 0FH
        MOV AH,0
        ADD AX,AX                 ;形成位移量
        MOV BX, OFFSET TAB        ;取转移表首地址
        ADD BX,AX                 ;形成转移表的偏移地址
        JMP BX
TAB:    JMP L0                    ;转移表
        JMP L1
        JMP L2
        JMP L3
        JMP L4
        JMP L5
        JMP L6
        JMP L7
        JMP L8
        JMP L9
L0:     MOV DL, 'A'
        JMP DISP
L1:     MOV DL, 'B'
        JMP DISP
L2:     MOV DL, 'C'
        JMP DISP
L3:     MOV DL, 'D'
        JMP DISP
L4:     MOV DL, 'E'
        JMP DISP
```

```
L5:     MOV DL, 'F'
        JMP DISP
L6:     MOV DL, 'G'
        JMP DISP
L7:     MOV DL, 'H'
        JMP DISP
L8:     MOV DL, 'I'
        JMP DISP
L9:     MOV DL, 'J'
DISP:   MOV AH,02H                  ;2 号功能调用,显示输出
        INT 21H
EXIT:   MOV AH,4CH
        INT 21H
CODE      ENDS
        END   START
```

(3) 逻辑分解法:

```
CODE   SEGMENT
     ASSUME   CS:CODE
START: MOV AH, 01H
        INT 21H                     ;1 号功能调用,键盘输入
        CMP AL, 30H
        JZ L0
        CMP AL, 31H
        JZ L1
        CMP AL, 32H
        JZ L2
        CMP AL, 33H
        JZ L3
        CMP AL, 34H
        JZ L4
        CMP AL, 35H
        JZ L5
        CMP AL, 36H
        JZ L6
        CMP AL, 37H
        JZ L7
        CMP AL, 38H
        JZ L8
        CMP AL, 39H
        JZ L9
        JMP EXIT
L0:     MOV DL, 'A'
        JMP DISP
L1:     MOV DL, 'B'
        JMP DISP
L2:     MOV DL, 'C'
        JMP DISP
L3:     MOV DL, 'D'
        JMP DISP
```

```
L4:     MOV DL, 'E'
        JMP DISP
L5:     MOV DL, 'F'
        JMP DISP
L6:     MOV DL, 'G'
        JMP DISP
L7:     MOV DL, 'H'
        JMP DISP
L8:     MOV DL, 'I'
        JMP DISP
L9:     MOV DL, 'J'
DISP:   MOV AH,02H                  ;2 号功能调用,显示输出
        INT 21H
EXIT:   MOV AH,4CH
        INT 21H
CODE      ENDS
        END   START
```

9.3 实验内容

【实验目的】 了解转移指令的一般用法,特别是条件转移指令与标志位的关系,掌握分支程序设计的一般方法。

【实验 9-1】 编程实现如下分段函数的计算,X 和 Y 均为字节变量。要求:先画出程序流程图,再编程。上机调试时,X 分别取正数、零、负数不同的值,测试运行结果的正确性,写出实验结果分析。

$$Y=\begin{cases}(X+5)/2, & X>0\\0, & X=0\\4X, & X<0\end{cases}$$

参考符号函数的方法即可实现。程序如下:

```
DSEG    SEGMENT
  X   DB   -5
  Y   DB   ?
DSEG    ENDS
CSEG    SEGMENT
    ASSUME   CS:CSEG, DS:DSEG
START: MOV   AX, DSEG
        MOV   DS,AX
        MOV   AL,X
        CMP   AL, 0
        JGE   POS
        SAL AL,1                ;X*2
        SAL AL,1                ;X*4
        JMP   DONE
POS:    JE   ZERO
        ADD AL,5                ;X+5
        SAR AL,1                ;(X+5)/2
```

```
        JMP   DONE
ZERO:   MOV   AL, 0
DONE:   MOV   Y, AL
        MOV   AH,4CH
        INT   21H
CSEG    ENDS
    END START
```

对于分支结构程序,运行程序时的测试用例要选取几个不同的值,以保证各个分支至少能执行一次,本例不妨 X 的取值为 -5、5、0 时,各执行一次程序,查看运行结果的正确性。这是分支结构程序与顺序结构程序在测试时的主要区别。

【实验 9-2】 在数据段中定义了两个长度为 30 的字符串 STR1 和 STR2,用串操作指令比较两个字符串是否相等。若两串相等,则显示输出字符'Y';若不相等,则记录下不相等单元的逻辑地址,分别存入 NO_ADDR1 和 NO_ADDR2 单元中。编程实现,并上机调试,写出实验过程和结果分析。

分析: 可以用 REPE CMPSB 指令,当出现两串不相等时,标志位 ZF=0,这时对应单元有不等之处,记录下存储单元逻辑地址。

程序如下:

```
DATA SEGMENT
   STR1 DB 'STRING1 IS A STRING1'
   STR2 DB 'STRING1 IS B STRING1'
   COUNT EQU $ - STR2
   NO_ADDR1 DW ?
   NO_ADDR2 DW ?
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       MOV ES,AX
       LEA SI,STR1
       LEA DI,STR2
       MOV CX,COUNT
       CLD
       REPE CMPSB
       JE YES                  ;完全相等,转 YES 处
       DEC SI                  ;不相等,记录下偏移地址
       DEC DI
       MOV NO_ADDR1,SI
       MOV NO_ADDR2,DI
       JMP EXIT
YES:   MOV DL,'Y'
       MOV AH,2
       INT 21H
EXIT:  MOV AH,4CH
       INT 21H
CODE ENDS
       END START
```

习　　题

1．定义三个字变量 A1、A2、MAX，比较 A1、A2 中带符号数的大小，将较大者送入 MAX 单元，编写程序并上机调试通过。

2．编写程序，根据下列原则判断 X、Y 字节变量中的数据，并为 FLAG 字节变量赋值。要求上机调试通过，并说明如何选取测试数据。

（1）若两个数都是偶数，则将 FLAG 置“2”；

（2）若两个数都是奇数，则将 FLAG 置“1”；

（3）若两个数为一奇一偶，则将 FLAG 置“0”。

3．定义三个字节变量 X、Y、Z，比较 X、Y 中带符号数的大小。当 $X>Y$ 时，将 Z 置“1”；当 $X<Y$ 时，将 Z 置“-1”；当 $X=Y$ 时，将 Z 清“0”，编写程序并上机调试通过。

4．阅读如下程序，完成：

（1）程序执行后，FLAG 单元的内容是什么？

（2）如果 X 的值定义为 0，则 FLAG 单元的内容是什么？

（3）该程序完成什么功能？

```
DATA SEGMENT
   X      DB 17H
   FLAG   DB ?
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV AL,X
        JGE NEXT
        MOV FLAG,0
        JMP EXIT
NEXT:   MOV FLAG,1
EXIT:   MOV AH,4CH
    INT 21H
CODE ENDS
    END START
```

第 10 章

循环程序设计

在串操作指令中，介绍了重复操作前缀指令，使重复前缀后面的串操作指令重复执行。实际应用中，也经常会出现一段程序需要重复执行若干次的情况，这就需要用循环结构来实现。本章主要介绍循环控制指令和循环程序设计，要求掌握循环程序的基本结构、程序设计及上机调试方法。

10.1 循环控制指令

循环控制指令主要包括：LOOP，LOOPZ/LOOPE，LOOPNZ/LOOPNE。其共同特点：在 CX 中存放循环次数；先执行 CX←CX－1，然后检查是否满足测试条件，若满足，则循环；若不满足，则退出循环，顺序执行下一条指令。这些循环控制指令都不影响状态标志位。

1. LOOP 指令

格式：LOOP　短标号

测试条件：(CX)≠0。

LOOP 指令要求，将 CX 作为计数寄存器，执行该指令时，其隐含操作是 CX 内容先减 1，然后判断 CX 内容是否为零。若(CX)≠0，则转到 LOOP 指令中指示的标号处，继续执行循环体；若(CX)＝0，则退出循环，顺序执行 LOOP 下一条指令。

2. LOOPZ/LOOPE 指令

格式：LOOPZ (或 LOOPE)　短标号

测试条件：ZF＝1 且(CX)≠0。

执行该指令时，其隐含操作是 CX 内容先减 1，仅当 ZF＝1 且(CX)≠0 时，即当规定的循环次数未完成时，必须满足为零或相等时，才转到 LOOPZ / LOOPE 指令中指示的标号处，继续执行循环体；当(CX)＝0 时或者出现不相等(或者不为零)时，使标志位 ZF＝0，则退出循环，顺序执行下一条指令。

3. LOOPNZ/LOOPNE 指令

格式：LOOPNZ (或 LOOPNE)　短标号

测试条件：ZF＝0 且(CX)≠0。

执行该指令时，其隐含操作是 CX 内容先减 1，仅当 ZF＝0 且(CX)≠0 时，即当规定的循环次数未完成时，必须满足不为零或不相等时，才转到 LOOPNZ/LOOPNE 指令中指示的标号处，继续执行循环体；当(CX)＝0 时或者出现相等(或者为零)时，使标志位 ZF＝1，则退出循环，顺序执行下一条指令。

10.2 循环程序结构及应用举例

循环程序结构一般分为三个部分：循环的初始化、循环体、循环控制部分。

1. 循环的初始化

设置循环的初始值，如设置循环计数器初值，或终止条件，或地址初值等信息。

2. 循环体

循环体包括循环的工作部分及修改部分。循环的工作部分，是为完成某种特定功能而设计的程序段，即重复操作的程序段；修改部分主要是循环参数的修改，如操作数地址指针的修改，为下一次循环做好准备。组成循环程序的工作部分可以是顺序结构、分支结构，也可以是一个循环结构。在循环体中又包含有循环结构，称为多重循环。

3. 循环控制部分

循环控制部分用于控制重复执行的次数，一般是检测循环结束条件。当循环结束条件不满足时，则继续重复执行循环体；当循环结束条件满足时，退出循环，执行循环结构的后续语句。循环控制部分主要由循环指令、条件转移指令等来实现。

典型的循环结构有两种形式："先判断，后执行"和"先执行，后判断"。图 10-1(a)是"先判断，后执行"，循环次数可能为零。图 10-1(b)是"先执行，后判断"，至少执行一次循环体。

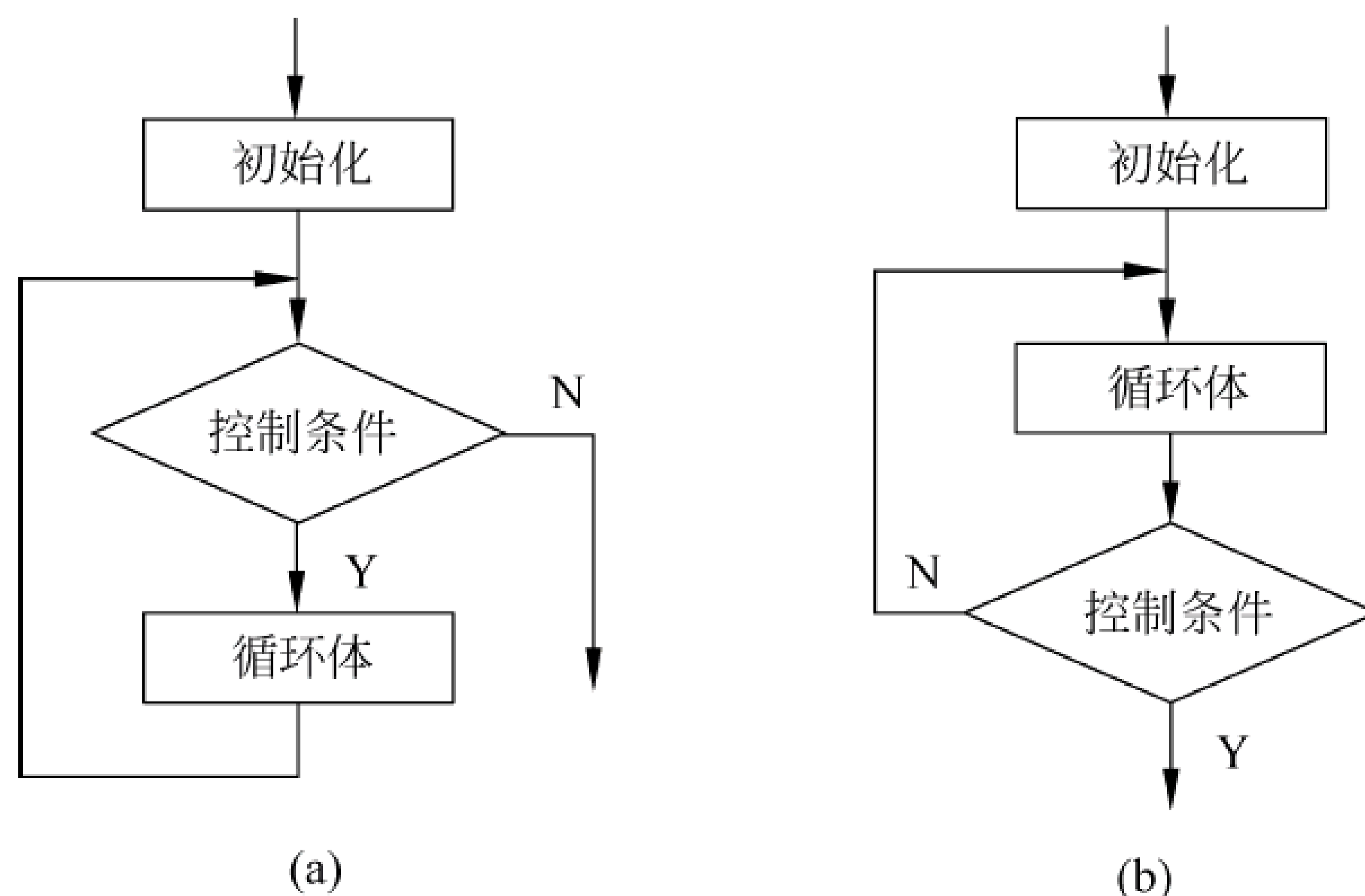


图 10-1 典型的循环结构

【例 10-1】 从 STR1 为起始地址的 30 个字符，依次传送到以 STR2 为起始地址的连续字节存储单元中。设 STR1 和 STR2 两个存储区域不发生重叠，请用不同寻址方式编写循环程序。如果 STR2 按逆序存储字符串，如何实现？

解析：

(1) 采用寄存器间接寻址方式，用 SI 作为指向 STR1 存储区的地址指针，DI 为指向 STR2 存储区的地址指针。

程序如下：

```
DSEG   SEGMENT
STR1   DB 'ABCDEFGHIJKLMNOPQRSTUVW'
       DB 'XYZ0123'
```

```
STR2   DB 30 DUP(?)
DSEG   ENDS
CSEG   SEGMENT
     ASSUME  DS:DSEG, CS:CSEG
START: MOV   AX,DSEG
       MOV   DS,AX
       MOV   SI,OFFSET  STR1          ;设置源地址指针
       MOV   DI,OFFSET  STR2          ;设置目标地址指针
       MOV   CX,30                    ;设置循环次数
NEXT:  MOV   AL,[SI]                  ;循环体
       MOV   [DI],AL                  ;字符传送
       INC   SI                       ;修改地址指针
       INC   DI
       LOOP  NEXT                     ;循环控制
       MOV   AH,4CH
       INT   21H
CSEG   ENDS
       END   START
```

（2）采用寄存器相对寻址方式。

程序如下：

```
DSEG   SEGMENT
STR1   DB 'ABCDEFGHIJKLMNOPQRSTUVW'
       DB 'XYZ0123'
STR2   DB 30 DUP(?)
DSEG   ENDS
CSEG   SEGMENT
     ASSUME  DS:DSEG,CS:CSEG
START: MOV   AX,DSEG
       MOV   DS,AX
       MOV   SI,0
       MOV   CX,30
NEXT:  MOV   AL,STR1[SI]              ;循环体
       MOV   STR2[SI],AL              ;字符传送
       INC   SI                       ;修改地址指针
       LOOP  NEXT
       MOV   AH,4CH
       INT   21H
CSEG   ENDS
       END   START
```

（3）采用基址变址寻址方式。

程序如下：

```
DSEG   SEGMENT
STR1   DB 'ABCDEFGHIJKLMNOPQRSTUVW'
       DB 'XYZ0123'
STR2   DB 30 DUP(?)
DSEG   ENDS
CSEG   SEGMENT
```

```
    ASSUME  DS:DSEG,CS:CSEG
START: MOV  AX,DSEG
       MOV  DS,AX
       MOV  BX,OFFSET STR1
       MOV  SI,0
       MOV  CX,30
NEXT:  MOV  AL,[BX][SI]            ;循环体
       MOV  30[BX][SI],AL          ;字符传送
       INC  SI                     ;修改地址指针
       LOOP  NEXT
       MOV  AH,4CH
       INT  21H
CSEG  ENDS
       END  START
```

(4) 如果 STR2 按与源字符串相反的顺序存储,即先将 STR1 的最后一个字符,传送到 STR2 的第一个存储单元作为首字符,依次类推,直至将 STR1 的首个字符,传送到 STR2 的最后一个存储单元。

程序如下:

```
DSEG  SEGMENT
STR1  DB 'ABCDEFGHIJKLMNOPQRSTUVW'
      DB 'XYZ0123'
STR2  DB 30 DUP(?)
DSEG  ENDS
CSEG  SEGMENT
    ASSUME  DS:DSEG,CS:CSEG
START: MOV  AX,DSEG
       MOV  DS,AX
       MOV  SI,OFFSET  STR1
       MOV  DI,OFFSET  STR2
       MOV  CX,30
NEXT:  MOV  AL,[SI + 29]           ;循环体
       MOV  [DI],AL                ;字符传送
       DEC  SI                     ;修改地址指针
       INC  DI
       LOOP  NEXT
       MOV  AH,4CH
       INT  21H
CSEG  ENDS
       END  START
```

关于循环程序的调试(以例 10-1 中采用寄存器间接寻址方式实现为例),在集成实验环境中,选择"运行"|"DEBUG 调试",进入调试环境,一般采用 T 命令或 P 命令跟踪执行,查看每一步的执行情况。具体操作方法如下:

(1) 单步执行时,在执行完段寄存器装填后,就可以用 D 命令查看数据段定义的情况,运行结果如下:

```
-T
AX = 13C5  BX = 0000  CX = 005A  DX = 0000  SP = 0000  BP = 0000  SI = 0000  DI = 0000
```

```
DS=13B5   ES=13B5   SS=13C5   CS=13C9   IP=0003    NV UP EI PL NZ NA PO NC
13C9:0003 8ED8              MOV       DS,AX
-T
AX=13C5   BX=0000   CX=005A   DX=0000   SP=0000   BP=0000   SI=0000   DI=0000
DS=13C5   ES=13B5   SS=13C5   CS=13C9   IP=0005    NV UP EI PL NZ NA PO NC
13C9:0005 BE0000            MOV       SI,0000
-D DS:0
13C5:0000  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50   ABCDEFGHIJKLMNOP
13C5:0010  51 52 53 54 55 56 57 58-59 5A 30 31 32 33 00 00   QRSTUVWXYZ0123..
13C5:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0040  B8 C5 13 8E D8 BE 00 00-BF 1E 00 B9 1E 00 8A 04   ................
13C5:0050  88 05 46 47 E2 F8 B4 4C-CD 21 00 00 00 00 00 00   ..FG..L.!.......
13C5:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

(2) 单步执行到 LOOP 语句时，如果想跟踪下一循环，可用 T 命令；如果循环次数较多，没有必要跟踪每一次循环过程，就可以用 P 命令执行 LOOP 语句。

(3) 执行 LOOP 语句后，数据段的存储情况可以用 D 命令查看。可以看到，STR1 的 30 个字符已经复制到 STR2 处，完成了字符串的传送。运行结果如下：

```
-P
AX=1341   BX=0000   CX=001E   DX=0000   SP=0000   BP=0000   SI=0001   DI=001F
DS=13C5   ES=13B5   SS=13C5   CS=13C9   IP=0014    NV UP EI PL NZ NA PO NC
13C9:0014 E2F8              LOOP      000E
-P
AX=1333   BX=0000   CX=0000   DX=0000   SP=0000   BP=0000   SI=001E   DI=003C
DS=13C5   ES=13B5   SS=13C5   CS=13C9   IP=0016    NV UP EI PL NZ NA PE NC
13C9:0016 B44C              MOV       AH,4C
-D DS:0
13C5:0000  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50   ABCDEFGHIJKLMNOP
13C5:0010  51 52 53 54 55 56 57 58-59 5A 30 31 32 33 41 42   QRSTUVWXYZ0123AB
13C5:0020  43 44 45 46 47 48 49 4A-4B 4C 4D 4E 4F 50 51 52   CDEFGHIJKLMNOPQR
13C5:0030  53 54 55 56 57 58 59 5A-30 31 32 33 00 00 00 00   STUVWXYZ0123....
13C5:0040  B8 C5 13 8E D8 BE 00 00-BF 1E 00 B9 1E 00 8A 04   ................
13C5:0050  88 05 46 47 E2 F8 B4 4C-CD 21 00 00 00 00 00 00   ..FG..L.!.......
13C5:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

(4) 如果运行结果不正确，需要找出问题所在，然后修改程序，重新汇编、运行，直至没有错误为止。按照这个方法，可以查看其他几种实现方式的字符串传送的结果。

【例 10-2】 对一组字节无符号数求最大值。假设一组字节无符号数存储在 BUFFER 开始的单元中，分别为：00H，12H，3BH，43H，60H，0CH，8AH，0ABH，37H，0FFH，32H，47H。

(1) 找出最大数，并将其存储在 MAX 单元中。

(2) 找出最大数，并将其以十六进制形式显示在屏幕上。

解析：

(1) 找最大数方法：将第一个数送 AL；AL 依次与下一个数比较，并保留较大的数；最后，AL 中的内容即为最大数，送入 MAX 单元。

程序如下：

```
DATA    SEGMENT
  BUFFER     DB 00H,12H,3BH,43H,60H,0CH
             DB 8AH,0ABH,37H,0FFH,32H,47H
  COUNT      EQU  $ - OFFSET BUFFER
  MAX        DB  ?
DATA    ENDS
CODE    SEGMENT
   ASSUME  CS:CODE, DS:DATA
START:MOV AX, DATA
      MOV DS, AX
      MOV SI,OFFSET BUFFER
      MOV AL,[SI]              ;第一个数送入 AL
      INC SI
      MOV CX,COUNT - 1
CMPARE: CMP AL,[SI]            ;AL 与下一个数比较
        JA NEXT                ;若大于,则转
        MOV AL,[SI]            ;否则,把较大者送入 AL
NEXT:  INC SI
      LOOP CMPARE
      MOV MAX,AL               ;最大数送入 MAX
      MOV AH,4CH
      INT 21H
CODE    ENDS
   END   START
```

(2) 若把最大数以十六进制形式显示在屏幕上，需要用 2 号功能调用。十六进制数转换成 ASCII 码方法：当十六进制数小于 0AH 时，+30H 即转换成 ASCII 码；十六进制数大于等于 0AH 时，+37H 即转换成 ASCII 码。

程序如下：

```
DATA    SEGMENT
  BUFFER  DB 00H,12H,3BH,43H,60H,0CH
          DB 8AH,0ABH,37H,0FFH,32H,47H
  COUNT  EQU  $ - OFFSET BUFFER
DATA    ENDS
CODE    SEGMENT
   ASSUME  CS:CODE, DS:DATA
START: MOV AX, DATA
       MOV DS, AX
       MOV SI,OFFSET BUFFER
       MOV AL,[SI]               ;第一个数送入 AL
       INC SI
       MOV CX,COUNT - 1
CMPARE: CMP AL,[SI]              ;AL 与下一个数比较
```

```
        JA NEXT                      ;若大于,则转
        MOV AL,[SI]                  ;否则,把较大者送入 AL
NEXT:   INC SI
        LOOP CMPARE
        MOV   BL,AL                  ;最大数已经在 AL 中,送入 BL 暂存
        MOV   DL,AL                  ;最大数 AL 送入 DL
        MOV   CL,4
        SHR   DL,CL                  ;将 DL 的高四位变成低四位,高四位为零
        CMP   DL,0AH                 ;与 0AH 比较
        JB   L1                      ;若小于,则转
        ADD DL, 7                    ;否则, + 7
L1:     ADD   DL, 30H                ; + 30H
        MOV AH, 2
        INT   21H                    ;输出十六进制数的高位
        MOV   DL, BL                 ;最大数送入 DL
        AND   DL,0FH                 ;屏蔽高四位
        CMP DL, 0AH                  ;与 0AH 比较
        JB   L2
        ADD DL, 7
L2:     ADD   DL, 30H
        MOV   AH, 2
        INT 21H                      ;输出十六进制数的低位
        MOV AH,4CH
        INT 21H
CODE    ENDS
   END   START
```

【例 10-3】 在数据段以 BUFFER 为首地址的存储区中,存放了 *N* 个非零字节数据。编程实现:分别统计正数和负数的个数送到 PLUS 和 MINUS 字节单元中。

对于正负数的判定,在不改变数值的情况下,一般有以下五种方法来测试 AL 中的带符号数:

```
(1) TEST   AL,AL                 ;可用 AND,OR 代替 TEST
     JS      L1                  ;为负转 L1
     ⋮                           ;否则为正
(2) TEST   AL,80H                ;测试最高位 D7 = 1
    JNZ   L1                     ;为负数转 L1
     ⋮                           ;否则为正
(3) XOR   AL,0                   ;跟 0 异或结果不变
     JS     L1                   ;为负数转 L1
     ⋮                           ;否则为正
(4) ADD    AL,0                  ;亦可用 SUB AL,0
     JS     L1                   ;为负数转 L1
     ⋮                           ;否则为正
(5) SHL   AL,1                   ;左移 1 位,最高位进入 CF
     JC  L1                      ;为负数转 L1
     ⋮                           ;否则为正
```

下面采用第(1)种方法,编程实现正负数的判定。程序如下:

```
DATA SEGMENT
   BUFFER   DB   -110,26,35,-46,3,-88,-46,38,67,20
   COUNT    EQU  $ - BUFFER
   PLUS     DB   0
   MINUS    DB   0
 DATA  ENDS
CODE  SEGMENT
   ASSUME  CS:CODE, DS:DATA
START:MOV   AX,DATA
      MOV   DS,AX
      LEA   BX,BUFFER
      MOV   CX,COUNT          ;设置计数器
NEXT: MOV   AL,[BX]
      TEST  AL,AL             ;生成状态标志位 SF 等
      JNS   L1                ;非负转 L1
      INC   MINUS             ;统计负数个数
      JMP   L2
L1:   INC   PLUS              ;统计正数个数
L2:   INC   BX
      LOOP NEXT
      MOV   AH,4CH
      INT   21H
CODE  ENDS
    END  START
```

程序运行后，单步执行，在 DEBUG 下查看运行结果如下。

```
-P
AX=1392  BX=0000  CX=000A  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=001C   NV UP EI PL NZ NA PO NC
13C6:001C 43              INC       BX
-P
AX=1392  BX=0001  CX=000A  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=001D   NV UP EI PL NZ NA PO NC
13C6:001D E2ED            LOOP      000C
-P
AX=1314  BX=000A  CX=0000  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=001F   NV UP EI PL NZ NA PE NC
13C6:001F B44C            MOV       AH,4C
-D DS:0
13C5:0000  92 1A 23 D2 03 A8 D2 26-43 14 06 04 00 00 00 00   ..#....&C.......
13C5:0010  B8 C5 13 8E D8 8D 1E 00-00 B9 0A 00 8A 07 84 C0   ................
13C5:0020  79 06 FE 06 0B 00 EB 04-FE 06 0A 00 43 E2 ED B4   y..........C...
13C5:0030  4C CD 21 00 00 00 00 00-00 00 00 00 00 00 00 00   L.!.............
13C5:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

从数据存储情况可以看出，(13C5：000A)＝06H，是正数的个数；(13C5：000B)＝

04H,是负数的个数。

【例 10-4】 在 BUFFER 开始的存储区定义一个以'$'为结束标志的字符串(串长取值范围 0～255),统计该字符串的长度(不包含结束标志'$'),并将结果存放在 STRLEN 字节单元中,并将串长以十进制数形式显示输出。

前面几个例题循环次数是已知的,采用的是"先执行,后判断"的循环结构。本例不同,在测试字符串长度时,循环次数事先不能确定,只能根据结束标志'$'来控制循环,这时应采用"先判断,后执行"的循环结构,这种循环控制方法又称条件控制法。

测试字符串长度时,用存储单元 STRLEN 作为计数器,循环检测字符,当遇到'$'时,结束循环,这时 STRLEN 单元中的值为二进制数的串长。

如果要显示输出,需要将二进制数转换为十进制数。转换方法：对于一个字节的无符号二进制数,可以用该数除以 100,商为百位；余数再除以 10,商为十位,余数为个位。

显示输出时,需要将十进制数转换为 ASCII 码。转换方法：对于 0～9 的十进制数字,加上 30H 后,即变为对应的数字字符的 ASCII 码,利用 2 号 DOS 功能调用显示输出。

程序如下：

```
DATA SEGMENT
  BUFFER DB 'ABCDEFGHIJKLMNOP123456789$'
  STRLEN DB 0
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       LEA SI,BUFFER
NEXT:  MOV AL,[SI]
       CMP AL,'$'
       JZ DISP2
       INC SI
       INC BYTE PTR STRLEN
       JMP NEXT
DISP2: MOV AH,0
       MOV AL,STRLEN
       MOV CL,100
       DIV CL              ;除以 100,商为百位数
       MOV BL,AH           ;余数在 AH 中,送 BL
       CMP AL,0            ;商和 0 比较,若为 0 则不显示
       JZ DISP1
       MOV DL,AL           ;显示百位数
       ADD DL,30H
       MOV AH,2
       INT 21H
DISP1: MOV AL,BL           ;处理十位数
       MOV AH,0
       MOV CL,10
       DIV CL              ;商为十位,余数为个位
       MOV BL,AH           ;余数送 BL
       CMP AL,0
```

```
        JZ DISP0
        MOV DL,AL           ;显示十位
        ADD DL,30H
        MOV AH,2
        INT 21H
DISP0:  MOV DL,BL           ;显示个位
        ADD DL,30H
        MOV AH,2
        INT 21H
EXIT:   MOV AH,4CH
        INT 21H
CODE    ENDS
        END START
```

在集成实验环境中，利用 DEBUG 运行该程序，其输出结果是 25（十六进制数为 19），输出结果及存储单元内容如下：

```
-P
DS=13B5  ES=13B5  SS=13C5  CS=13C7  IP=0003   NV UP EI PL NZ NA PO NC
13C7:0003 8ED8            MOV     DS,AX
-P
AX=13C5  BX=0000  CX=0061  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C7  IP=0005   NV UP EI PL NZ NA PO NC
13C7:0005 8D360000        LEA     SI,[0000]                   DS:0000=4241
-P
AX=13C5  BX=0000  CX=0061  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C7  IP=0009   NV UP EI PL NZ NA PO NC
13C7:0009 8A04            MOV     AL,[SI]                     DS:0000=41
-G
25
Program terminated normally
-D DS:0
13C5:0000  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50   ABCDEFGHIJKLMNOP
13C5:0010  31 32 33 34 35 36 37 38-39 24 19 00 00 00 00 00   123456789$......
13C5:0020  B8 C5 13 8E D8 8D 36 00-00 8A 04 3C 24 74 07 46   ......6....<$t.F
13C5:0030  FE 06 1A 00 EB F3 B5 00-B3 0A 8A 0E 1A 00 80 F9   ................
13C5:0040  0A 73 02 EB 06 2A CB FE-C5 EB F3 80 C1 30 80 C5   .s...*.......0..
13C5:0050  30 B4 02 8A D5 CD 21 B4-02 8A D1 CD 21 B4 4C CD   0.....!.....!.L.
13C5:0060  21 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   !...............
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

10.3 多重循环

如果循环结构的循环体中又出现一个完整的循环结构，就构成多重循环结构。一般常用的有二重循环和三重循环。循环层数越多，运行时间越长，程序越复杂。在使用多重循环时需要注意以下几点：

（1）内循环必须完整地包含在外循环体内，不允许内外循环交叉嵌套。

（2）可以从内循环中直接跳转到外循环，或者跳出外循环；不允许从外循环跳入内循环。

（3）防止出现“死循环”。无论是内循环还是外循环，千万不能让循环返回到初始化部分，严格按照循环结构的控制流程。

图 10-2 表示的是二重循环结构，内层初始化、内层循环体和内层循环控制部分构成了一个完整的内层循环；内层循环与外层程序段 1 和外层程序段 2 一起构成了外层循环的循环体。

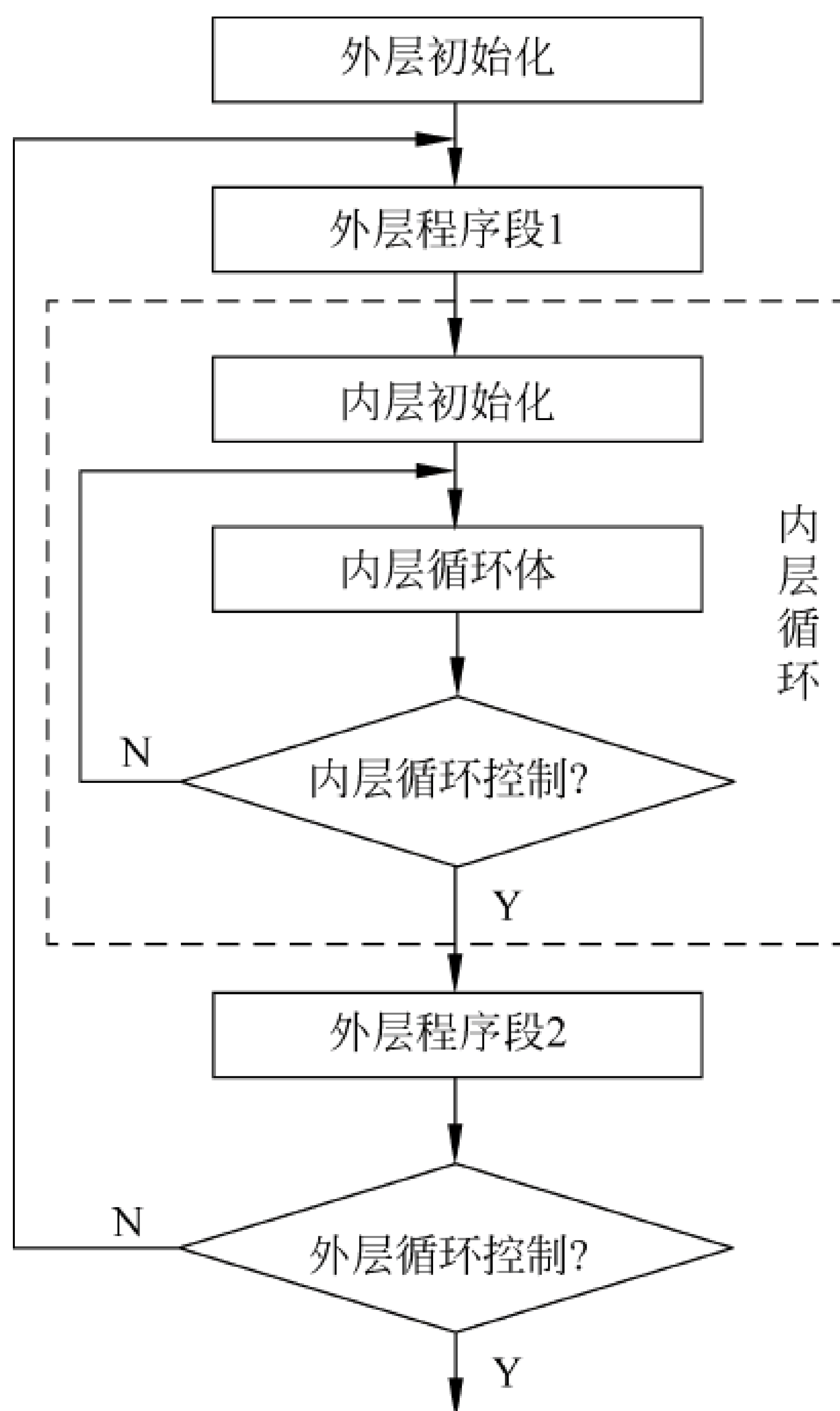


图 10-2　二重循环结构图

【例 10-5】　用选择排序方法，实现对一组字节无符号数按由小到大的顺序排列，排序结果仍存放在原地址处。

多重循环的典型应用就是排序问题，排序算法有选择排序、冒泡排序、插入排序等。这里采用选择排序。基本思想：按照升序排序的要求，首先对序列进行扫描，从中找到最小数，将其与第一个数互换，放在首地址处；然后，对剩下的元素进行扫描，找到其中最小数，将其与第二个数互换，放在第二个元素的位置，即第二个存储单元处；依次类推，直至完成所有元素的排序。

这是一个二重循环的问题，程序设计时用 CX 做外层循环的计数器，SI 做外层循环的地址指针；用 DX 做内层循环的计数器，DI 做内层循环的地址指针。程序如下：

```
DATA SEGMENT
 BUFFER DB 10H,93H,15H,16H,80H,05H
        DB 56H,78H,37H,18H,56H,02H
        DB 2AH,20H,5FH,5DH,3EH,43H
  COUNT EQU $ - BUFFER
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       MOV SI,OFFSET BUFFER       ;外层地址指针 SI
       MOV CX,COUNT - 1           ;外层循环计数 CX
NEXT1: MOV DX,CX                  ;外层循环计数 DX
       MOV DI,SI                  ;内层地址指针 DI
       MOV AL,[DI]
NEXT2: INC DI
       CMP AL,[DI]                ;与下一个数比较
       JB NEXT3                   ;小于,转移
       XCHG AL,[DI]
NEXT3: DEC DX
       JNZ NEXT2                  ;内层循环控制
```

```
        MOV [SI],AL                 ;存最小数
        INC SI
        LOOP NEXT1
EXIT:   MOV AH,4CH
        INT 21H
CODE ENDS
    END START
```

在集成实验环境中,用 DEBUG 调试。单步执行完前两条语句,即完成了数据段寄存器的装填,这时可以查看到排序前数据的存储情况。然后,直接用 G 命令执行后,再次查看到数据的存储情况,这时就是排序后的结果了。运行结果如下:

```
-P
AX=13C5  BX=0000  CX=0044  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13C7  IP=0003   NV UP EI PL NZ NA PO NC
13C7:0003 8ED8            MOV     DS,AX
-P
AX=13C5  BX=0000  CX=0044  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C7  IP=0005   NV UP EI PL NZ NA PO NC
13C7:0005 BE0000          MOV     SI,0000
-D DS:0
13C5:0000  10 93 15 16 80 05 56 78-37 18 56 02 2A 20 5F 5D   ......Vx7.V.*_]
13C5:0010  3E 43 00 00 00 00 00 00-00 00 00 00 00 00 00 00   >C..............
13C5:0020  B8 C5 13 8E D8 BE 00 00-B9 11 00 8B D1 8B FE 8A   ................
13C5:0030  05 47 3A 05 72 02 86 05-4A 75 F6 88 04 46 E2 EB   .G:.r...Ju...F..
13C5:0040  B4 4C CD 21 00 00 00 00-00 00 00 00 00 00 00 00   .L.!............
13C5:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-G
Program terminated normally
-D DS:0
13C5:0000  02 05 10 15 16 18 20 2A-37 3E 43 56 56 5D 5F 78   ......*7>CVV]_x
13C5:0010  80 93 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0020  B8 C5 13 8E D8 BE 00 00-B9 11 00 8B D1 8B FE 8A   ................
13C5:0030  05 47 3A 05 72 02 86 05-4A 75 F6 88 04 46 E2 EB   .G:.r...Ju...F..
13C5:0040  B4 4C CD 21 00 00 00 00-00 00 00 00 00 00 00 00   .L.!............
13C5:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

10.4 实验内容

【实验目的】 理解循环结构的基本组成和程序设计的方法,掌握循环控制指令的用法,熟练使用 DEBUG 调试循环结构程序。

【实验 10-1】 在 BLOCK 单元开始的存储区中,连续存放着 30 个学生的某门课程的成绩,统计其中各分数段的学生数。要求 90~100 分、80~89 分、70~79 分、60~69 分及 60

分以下的学生数分别存放在 S9、S8、S7、S6 和 S5 中。

解析：统计各分数段学生人数时，分别用了五个计数器 S9、S8、S7、S6 和 S5，当出现大于 100 分的成绩时，属于无效数据，程序结束。程序设计思路可参考图 10-3 的流程图。在循环结构中，包含有多分支程序结构。

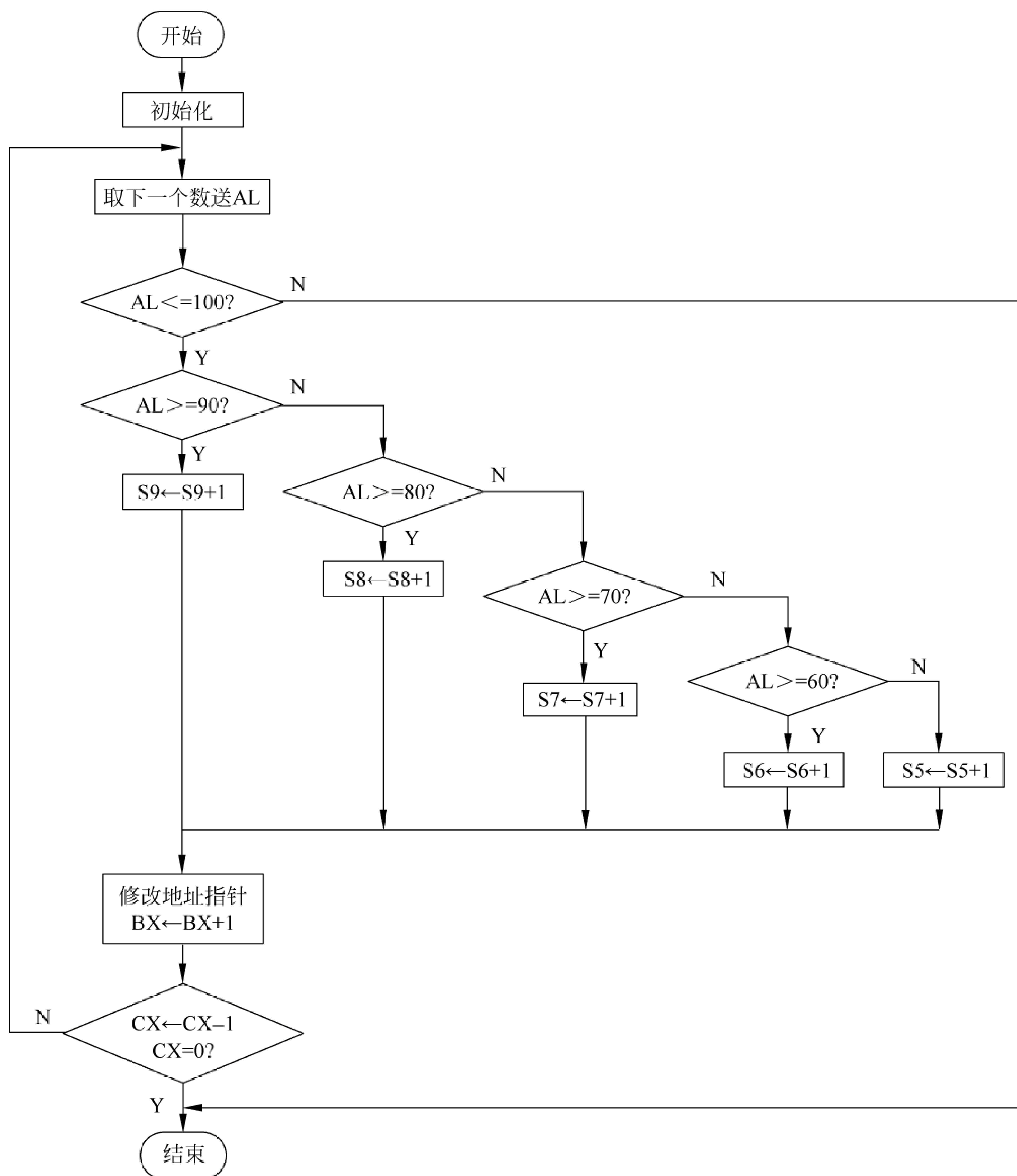


图 10-3　统计各分数段学生数的流程图

程序如下：

```
DATA SEGMENT
 BLOCK DB 40,50,60,78,84,98,100,89,79,69
       DB 95,59,70,80,90,62,81,47,73,85
       DB 66,68,76,74,83,87,91,62,86,88
```

```
    COUNT EQU $ - BLOCK
    S9  DB 0
    S8  DB 0
    S7  DB 0
    S6  DB 0
    S5  DB 0
DATA ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:DATA
START:MOV AX,DATA
      MOV DS,AX
      MOV BX,OFFSET BLOCK
      MOV CX,COUNT
NEXT: MOV AL,[BX]
      CMP AL,100
      JA EXIT
      CMP AL,90
      JAE L9
      CMP AL,80
      JAE L8
      CMP AL,70
      JAE L7
      CMP AL,60
      JAE L6
L5:   INC S5
      JMP L10
L6:   INC S6
      JMP L10
L7:   INC S7
      JMP L10
L8:   INC S8
      JMP L10
L9:   INC L10
L10:  INC BX
      LOOP NEXT
EXIT: MOV AH,4CH
      INT 21H
CODE ENDS
     END START
```

【实验 10-2】 在键盘上输入一个 3～9 的数字 N,则输出一个 N 行 N 列 * 号组成的方块。例如输入 3,则输出:

```
***
***
***
```

解析:利用 1 号 DOS 功能调用,读入数字 N 到 AL,判断是否为字符'3'～'9'。如果是,将其转换为二进制数送 CX,做循环计数器,控制输出;如果不是,则重新输入。该程序需要采用二重循环实现,其中关键点是循环计数器 CX 的使用。由于内层和外层都需要 CX 做

计数，因此这里使用了 BX 暂存数字 N 的 BCD 码，用 PUSH/POP 指令保护外层循环的 CX，使内层计数器和外层计数器互不影响。程序设计流程图如图 10-4 所示。

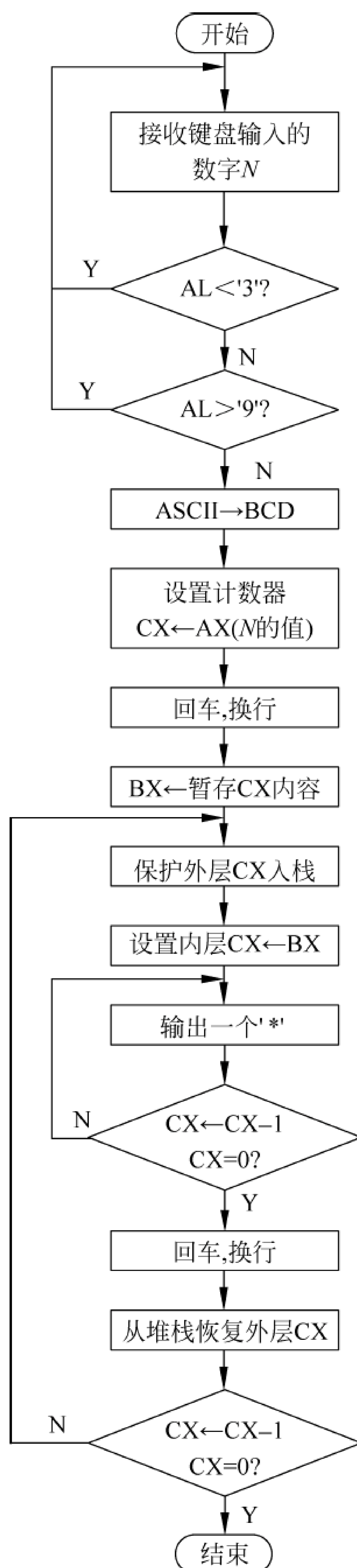


图 10-4　输出方块图形的流程图

程序如下：

```
CODE   SEGMENT
```

```
    ASSUME CS:CODE
START: MOV AH,1
       INT 21H
       CMP AL,'3'
       JB START
       CMP AL,'9'
       JA START
       SUB  AL,30H              ;ASCII to BCD
       MOV CL,AL
       MOV CH,0                 ;外层循环计数器设置,CX 为输入的数字 N
       MOV AH,2
       MOV DL,0DH
       INT 21H
       MOV DL,0AH
       INT 21H                  ;回车,换行
       MOV BX,CX
L2:    PUSH CX                  ;保护外层循环计数器
       MOV CX,BX                ;内层循环计数器设置
L1:    MOV DL,'*'
       MOV AH,02H
       INT 21H
       LOOP L1                  ;输出 *
       MOV DL,0DH
       INT 21H
       MOV DL,0AH
       INT 21H                  ;回车,换行
       POP CX                   ;恢复外层循环计数器
       LOOP L2
       MOV AH,4CH
       INT 21H
CODE   ENDS
    END START
```

习　题

1. 定义字变量 N 和 SUM,编程实现 SUM=1+2+3+…+N。

2. 在 BUFFER 开始的存储区定义一个以'$'为结束标志的字符串,统计该字符串中含有的数字字符的个数、大写字母的个数、小写字母的个数和其他字符的个数,将统计结果存放在内存单元中。

3. 将 BUFFER 开始的存储区中的 100 个字节型带符号数按正数和负数分开,分别送至同一个数据段的 BUFFER1 和 BUFFER2 两个缓冲区。

4. 用冒泡排序方法,实现对一组字节无符号数按由小到大的顺序排列,排序结果仍存放在原地址处。

5. 阅读如下程序,完成:

(1) 该程序完成什么功能?

(2) 程序执行后,SUM 单元的内容是什么?

```
DATA SEGMENT
   SUM DW ?
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        XOR AX,AX
        MOV CX,10
        MOV BX,2
NEXT:   ADD AX,BX
        ADD BX,2
        LOOP NEXT
        MOV SUM,AX
        MOV AH,4CH
        INT 21H
CODE ENDS
    END START
```

6．阅读如下程序，完成：

(1) 该程序完成什么功能？

(2) 程序执行后，BUFFER 单元的内容是什么？

```
DATA SEGMENT
   BUFFER  DB 'sql SERVER 123 * 89 abcd'
   COUNT   EQU $ - BUFFER
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV CX,COUNT
        LEA BX,BUFFER
LOP:    MOV AL,[BX]
        CMP AL,'A'
        JB NEXT
        CMP AL,'Z'
        JA NEXT
        ADD AL,20H
        MOV [BX],AL
NEXT:   INC BX
        LOOP LOP
        MOV AH,4CH
        INT 21H
CODE ENDS
    END START
```

7．阅读如下程序，完成：

(1) 该程序完成什么功能？

(2) 程序执行后，以 DA2 为首地址的前五个字节单元的内容是什么？

```
DATA SEGMENT
   DA1   DB '0123456789'
   DA2   DB 10 DUP(?)
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV CX,5
        MOV BX,5
        MOV SI,0
        MOV DI,0
NEXT:   MOV AL,[BX + SI + DA1]
        MOV [DI + DA2],AL
        INC SI
        INC DI
        LOOP NEXT
        MOV AH,4CH
        INT 21H
CODE ENDS
    END START
```

第 11 章

子程序

对于一个程序段,如果多次在程序的不同部分被重复使用,就可以把它分离出来,定义成一个子程序(或过程)。本章主要介绍子程序的定义及其调用,子程序的设计方法及参数传送方式。

11.1 子程序定义及其调用

1. 子程序定义格式

```
过程名  PROC    NEAR ( FAR )
   …
过程名  ENDP
```

说明:过程名是子程序的入口地址,命名方式与标号相同。类型属性为 NEAR 时,调用程序和子程序在同一代码段(段内调用);FAR 属性是指调用程序和子程序不在同一代码段(段间调用)。

2. 子程序调用

主程序利用 CALL 指令,将控制转移到过程,执行过程中的指令序列,称为子程序调用。CALL 指令不影响标志位。CALL 指令有以下四种格式:

1) 段内直接近调用

格式:CALL <过程名>

执行操作:PUSH (IP)

(IP)← (IP) + 16 位位移量

执行过程:保护断点,返回地址(即 CALL 指令的下一条指令的地址)推入堆栈;转子,转移到子程序的入口地址处执行。

2) 段内间接近调用

格式:CALL <寄存器或存储器>

执行操作:PUSH (IP)

(IP)← (EA)

执行过程:由指定的寄存器或存储单元的内容给出转向地址。

在近调用中,转向地址中只包含其偏移地址部分,段地址保持不变,即 CS 固定不变,只改变 IP 值。

3) 段间直接远调用

格式:CALL <过程名>

执行操作：PUSH (CS)
　　　　　PUSH (IP)
　　　　　(IP)←过程入口的位移量
　　　　　(CS)←过程所在段的段基值

执行过程：保护断点，返回地址（即 CALL 指令的下一条指令的地址，CS 和 IP 值）推入堆栈；转子，将子程序所在段的段基值送 CS，其入口地址的位移量送 IP，实现控制转移。

4）段间间接远调用

格式：CALL <存储器操作数>

执行操作：PUSH (CS)
　　　　　PUSH (IP)
　　　　　(IP)←(EA)
　　　　　(CS)←(EA+2)

执行过程：保护断点，返回地址（即 CALL 指令的下一条指令的地址，CS 和 IP 值）推入堆栈；转子，将指令中指定的双字存储器操作数的第一个字的内容送 IP，第二个字的内容送 CS，实现控制转移。

3. 子程序返回

过程执行完毕，利用 RET 指令返回到主程序继续执行。下面介绍 RET 指令的格式及功能。

格式：RET 或 RET <参数>

功能：子程序完成后，返回到调用该过程的 CALL 指令的下一条指令处继续执行。返回地址是 CALL 指令存入堆栈的指令地址。如果带有<参数>，则表示返回时从堆栈中舍弃的字节数。

返回操作实质：对段内近返回，栈顶一个字的内容弹出送入 IP；而对段间远返回，先弹出一个字送 IP；再弹出一个字送入 CS。

4. 子程序段内调用和返回的一般形式

```
CODE   SEGMENT
MAIN   PROC   FAR
         …
       CALL SUBP
         …
       RET
MAIN   ENDP

SUBP   PROC   NEAR
        …
       RET
SUBP   ENDP
CODE   ENDS
```

11.2　子程序设计

1. 确定子程序的入口参数和出口参数

在设计子程序时,一般都有一个子程序说明文件,其中最关键的是入口参数和出口参数,以便于其他人使用。入口参数要说明子程序运行中所需要的参数及存放位置,出口参数要说明子程序运行后的结果参数和存放位置。

2. 现场保护与恢复

子程序中使用的寄存器,如果同主程序中使用的寄存器发生冲突,应用先保护寄存器。为避免主程序和子程序在使用寄存器上发生冲突,可以将子程序中用到的寄存器内容先暂存在堆栈中,在子程序结束前,再恢复寄存器内容。

保存与恢复寄存器的工作可以在主程序中完成,也可以在子程序中完成。最常用的方法就是利用 PUSH/POP 指令。例如,对寄存器 AX、BX、CX、DX 进行保护与恢复,使用的程序段格式如下:

```
SUBP  PROC  NEAR
      PUSH  AX
      PUSH  BX
      PUSH  CX
      PUSH  DX
        …
        …
      POP  DX
      POP  CX
      POP  BX
      POP  AX
SUBP  ENDP
```

3. 确定参数传送方式

调用程序传递输入参数(或称入口参数)和子程序传递输出参数(或称出口参数)的过程,称为调用程序与子程序之间的参数传递。参数传送方式一般有三种:

(1) 寄存器:把输入参数放在寄存器中,子程序可以在指定寄存器中取出输入参数,子程序的输出参数也可以放在指定的寄存器中。由于寄存器数量有限,这种方法适用于参数较少情况。

(2) 存储器:在存储器中开辟一个约定的存储区,按照事先约定的次序存放输入参数和输出参数。这种方法适用于参数较多的情形。

(3) 堆栈:调用程序将输入参数压入堆栈,子程序从堆栈中弹出这些数据。输出参数也可以用类似的方法完成。这种方法同样适用于参数较多的情形。特别注意,从调用程序执行 CALL 指令进入子程序时,栈顶单元的内容是断点地址。

【例 11-1】 在键盘上输入一个 3～9 的数字 N,则输出一个 N 行 N 列 * 号组成的方块。要求编写如下子程序和调用程序:

(1) 实现输出一行 N 个 * 号,入口参数:BX←输出 * 号个数;出口参数:无。

(2) 实现回车换行功能,入口参数:无;出口参数:无。

(3) 编写调用程序,实现输出 N 行 N 列 * 号组成的方块。

程序如下:

```
CSEG  SEGMENT
    ASSUME  CS:CSEG
MAIN  PROC  FAR                     ;按主过程定义
START:     PUSH  DS
           SUB   AX,AX
           PUSH  AX
           MOV AH,1
           INT 21H
           CMP AL,'3'
           JB START
           CMP AL,'9'
           JA START
           SUB  AL,30H              ;ASCII to BCD
           MOV CL,AL
           MOV CH,0                 ;外层循环计数器设置,CX 为输入的数字 N
           CALL NEWLINE             ;回车,换行
           MOV BX,CX
L2:        PUSH CX                  ;保护外层循环计数器
           CALL OUTDSP
           CALL NEWLINE
           POP CX                   ;恢复外层循环计数器
           LOOP L2
           RET
;*************************************
;子程序名: NEWLINE
;功能: 回车换行
;入口参数: 无
;出口参数: 无
;占用寄存器: AX
;*************************************
NEWLINE    PROC   NEAR
           MOV AH,2
           MOV DL,0DH
           INT 21H
           MOV DL,0AH
           INT 21H
           RET
NEWLINE ENDP
;************************************
;子程序名: OUTDSP
;功能: 在屏幕上显示 N 个 * 号
;入口参数: BX←输出 * 号个数
;出口参数: 无
;占用寄存器: AX,DX
;************************************
OUTDSP  PROC  NEAR
           MOV CX,BX
```

```
L1:       MOV DL,'*'
          MOV AH,02H
          INT 21H
          LOOP L1
          RET
OUTDSP    ENDP
MAIN      ENDP
CSEG      ENDS
      END  START
```

关于子程序的调试，在集成实验环境中，选择“运行”|“DEBUG 调试”，进入调试环境，一般采用 T 命令或 P 命令跟踪执行，查看每一步的执行情况。当执行 CALL 语句时，如果执行 T 命令，则跟踪到相应的子程序内部单步执行；如果执行 P 命令，则不再跟踪到子程序内部，直接进到 CALL 语句的下一条语句处。

以例 11-1 为例，在如下的运行结果中，CALL 0026 就是程序中的 CALL NEWLINE 语句，这时执行 T 命令，则控制转向(13C5：0026)处，就是子程序 NEWLINE 的入口处。这时可以看到 SP 寄存器的内容减 2，栈顶单元存储的就是子程序的返回地址 0019，即 CALL 语句的下一条语句处的偏移地址。

```
DS=13B5  ES=13B5  SS=13C5  CS=13C5  IP=0010   NV UP EI NG NZ AC PE CY
13C5:0010 2C30           SUB      AL,30
-T
AX=0105  BX=0000  CX=003C  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13C5  IP=0012   NV UP EI PL NZ NA PE NC
13C5:0012 8AC8           MOV      CL,AL
-T
AX=0105  BX=0000  CX=0005  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13C5  IP=0014   NV UP EI PL NZ NA PE NC
13C5:0014 B500           MOV      CH,00
-T
AX=0105  BX=0000  CX=0005  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13C5  IP=0016   NV UP EI PL NZ NA PE NC
13C5:0016 E80D00         CALL      0026
-T
AX=0105  BX=0000  CX=0005  DX=0000  SP=FFFA  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13C5  IP=0026   NV UP EI PL NZ NA PE NC
13C5:0026 B402           MOV      AH,02
-D SS:FFFA
13C5:FFF0                          19 00 00 00 B5 13          ......
-
```

如果在 CALL 0026 处执行 P 命令，则不再跟踪到子程序的内部，子程序的调用与返回过程直接一步完成，下一条语句为“MOV BX,CX”。类似地，执行 INT 21H 指令，一般也都采用 P 命令。在 CALL 语句处执行 P 命令，运行结果如下：

```
AX=0105  BX=0000  CX=0005  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0000
DS=13B5  ES=13B5  SS=13C5  CS=13C5  IP=0014   NV UP EI PL NZ NA PE NC
13C5:0014 B500           MOV      CH,00
-T
```

```
AX = 0105  BX = 0000  CX = 0005  DX = 0000  SP = FFFC  BP = 0000  SI = 0000  DI = 0000
DS = 13B5  ES = 13B5  SS = 13C5  CS = 13C5  IP = 0016   NV UP EI PL NZ NA PE NC
13C5:0016 E80D00            CALL        0026
-P
AX = 020A  BX = 0000  CX = 0005  DX = 000A  SP = FFFC  BP = 0000  SI = 0000  DI = 0000
DS = 13B5  ES = 13B5  SS = 13C5  CS = 13C5  IP = 0019   NV UP EI PL NZ NA PE NC
13C5:0019 8BD9              MOV        BX,CX
-
```

【例 11-2】 编制一个简单的加密和解密程序，其功能为：

(1) 从键盘输入一位数字(0～9)，经加密后存入内存单元。从键盘输入数字由子程序KEYIN 完成。

(2) 将加密过的数据进行解密，并将解密后的数字在屏幕上显示出来，其显示可由子程序 DISP 完成。

假设数字 0～9 的加密和解密约定关系如下：

原始数字：0 1 2 3 4 5 6 7 8 9，即未经加密的数字；

密码数字：6 2 9 1 3 7 8 0 5 4，即对应的密码；

解密数字：7 3 1 4 9 8 0 5 6 2。

解析：程序首先完成加密操作，加密后的数字存储在 MIMA 单元，然后进行解密，解密后的数字显示输出。

程序如下：

```
DSEG  SEGMENT
MIMAB    DB  '6291378054'          ;可任意拟定
JMIMAB   DB  '7314980562'          ;根据 MIMA 拟定
MIMA     DB  ?                     ;存一位密码
DSEG  ENDS
CSEG  SEGMENT
       ASSUME  CS:CSEG,DS:DSEG
MAIN   PROC  FAR                   ;按主过程定义
START: PUSH  DS
       SUB   AX,AX
       PUSH  AX
       MOV   AX,DSEG
       MOV   DS,AX
       CALL  KEYIN
       MOV   BX,OFFSET MIMAB
       XLAT
       AND   AL,0FH
       MOV   MIMA,AL               ;存密码
       MOV   BX,OFFSET JMIMAB
       XLAT
       CALL  DISP
       RET
;*********************************************
;子程序名: KEYIN
;功能: 从键盘输入一个 0～9 数字,并转换为 BCD 码
;入口参数: 无
```

```
;出口参数: AL
;占用寄存器: AX
;*******************************************
KEYIN   PROC    NEAR
          MOV   AH,1
          INT   21H
          AND   AL,0FH
          RET
KEYIN   ENDP
;*******************************************
;子程序名: DISP
;功能: 将加密后的数据在屏幕上显示出来
;入口参数: AL
;出口参数: 无
;占用寄存器: AX,DX
;*******************************************
DISP   PROC   NEAR
       OR   AL,30H
       MOV   DL,AL
       MOV   AH,2
       INT   21H
       RET
DISP   ENDP
MAIN   ENDP
CSEG   ENDS
  END   START
```

【例 11-3】 分别利用存储器和堆栈传递参数的方法编写程序，实现对字节数组元素求和，并将结果保存在字变量 SUM 单元。

(1) 利用存储器传递参数，有直接存储单元传递和地址表两种方法。这里先采用直接存储单元传递参数，程序如下：

```
DATA SEGMENT
 ARRAY  DB 10,20,30,40,50,60,70,80,90
 COUNT  DW $ - ARRAY
 SUM    DW 0
DATA ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA
MAIN   PROC   FAR
START: PUSH DS
       SUB AX,AX
       PUSH AX
       MOV AX,DATA
       MOV DS,AX
       CALL SUM1                    ;调用子程序
       RET
MAIN   ENDP
;*************************************
;子程序名: SUM1
```

```
;功能：求字节数组元素之和
;入口参数：使用存储单元 ARRAY、COUNT 传递参数
;出口参数：运算结果在 SUM 单元
;占用寄存器：AX,CX,SI
;**************************************
SUM1    PROC NEAR
        PUSH AX                 ;保护现场
        PUSH CX
        PUSH SI
        XOR AX,AX
        LEA SI,ARRAY
        MOV CX,COUNT
ADDPRO:ADD AL,[SI]
        ADC AH,0
        INC SI
        LOOP ADDPRO
        MOV SUM,AX
        POP SI                  ;恢复现场
        POP CX
        POP AX
        RET
SUM1   ENDP
CODE ENDS
    END START
```

程序运行结果如下：

```
AX = 0000   BX = 0000   CX = 0038   DX = 0000   SP = FFFE   BP = 0000   SI = 0000   DI = 0000
DS = 13B5   ES = 13B5   SS = 13C5   CS = 13C6   IP = 0003   NV UP EI PL ZR NA PE NC
13C6:0003 50                 PUSH       AX
-P
AX = 0000   BX = 0000   CX = 0038   DX = 0000   SP = FFFC   BP = 0000   SI = 0000   DI = 0000
DS = 13B5   ES = 13B5   SS = 13C5   CS = 13C6   IP = 0004   NV UP EI PL ZR NA PE NC
13C6:0004 B8C513             MOV        AX,13C5
-P
AX = 13C5   BX = 0000   CX = 0038   DX = 0000   SP = FFFC   BP = 0000   SI = 0000   DI = 0000
DS = 13B5   ES = 13B5   SS = 13C5   CS = 13C6   IP = 0007   NV UP EI PL ZR NA PE NC
13C6:0007 8ED8               MOV        DS,AX
-P
AX = 13C5   BX = 0000   CX = 0038   DX = 0000   SP = FFFC   BP = 0000   SI = 0000   DI = 0000
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C6   IP = 0009   NV UP EI PL ZR NA PE NC
13C6:0009 E80100             CALL       000D
-P
AX = 13C5   BX = 0000   CX = 0038   DX = 0000   SP = FFFC   BP = 0000   SI = 0000   DI = 0000
DS = 13C5   ES = 13B5   SS = 13C5   CS = 13C6   IP = 000C   NV UP EI PL NZ NA PE NC
13C6:000C CB                 RETF
-D DS:0
13C5:0000  0A 14 1E 28 32 3C 46 50-5A 09 00 C2 01 00 00 00   ...(2<FPZ.......
13C5:0010  1E 2B C0 50 B8 C5 13 8E-D8 E8 01 00 CB 50 51 56   .+.P........PQV
13C5:0020  33 C0 8D 36 00 00 8B 0E-09 00 02 04 80 D4 00 46   3.6...........F
13C5:0030  E2 F8 A3 0B 00 5E 59 58-C3 00 00 00 00 00 00 00   .....^YX........
```

```
13C5:0040  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00   ................
13C5:0050  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00   ................
13C5:0060  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00   ................
13C5:0070  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00   ................
-
```

(2) 利用地址表传递参数方法实现。调用子程序前，要求把所有参数的地址送入地址表，然后把地址表的偏移量通过寄存器带进子程序，子程序从地址表中取出参数地址。

本例题中有三个参数，分别是数组元素(ARRAY)首地址、元素个数(COUNT)首地址、SUM 首地址。建立一个地址表 ADDR_TAB，如图 11-1 所示。

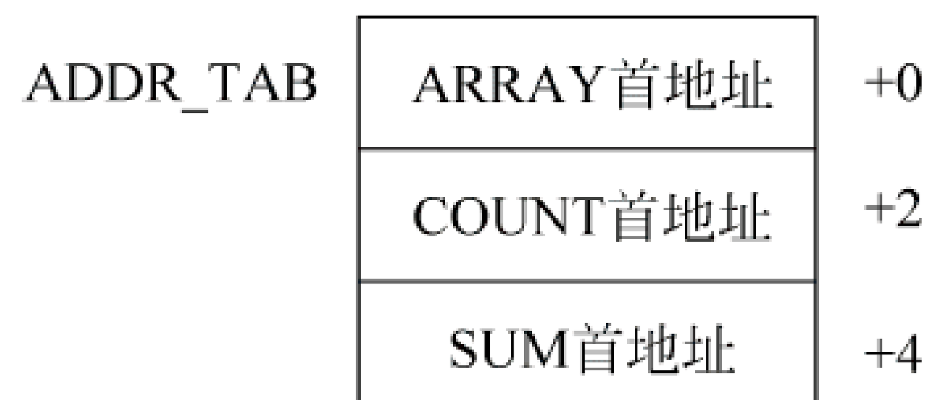


图 11-1　地址表传递参数示意图

程序如下：

```
DATA SEGMENT
 ARRAY DB 10,20,30,40,50,60,70,80,90
 COUNT DW  $ - ARRAY
 SUM   DW 0
 ADDR_TAB DW 3 DUP(?)
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
MAIN   PROC   FAR
START: PUSH   DS
       SUB    AX,AX
       PUSH   AX
       MOV    AX,DATA
       MOV    DS,AX
       MOV    ADDR_TAB,OFFSET ARRAY
       MOV    ADDR_TAB + 2,OFFSET COUNT
       MOV    ADDR_TAB + 4,OFFSET SUM
       LEA    BX,ADDR_TAB
       CALL   SUM2                    ;调用子程序
       RET
MAIN   ENDP
;**************************************************
;子程序名: SUM2
;功能: 求字节数组元素之和
;入口参数: BX←地址表首地址
;出口参数: 运算结果在 SUM 单元
;占用寄存器: AX,CX,SI,DI
;**************************************************
SUM2   PROC NEAR
       PUSH AX                        ;保护现场
       PUSH CX
       PUSH SI
       PUSH DI
       XOR  AX,AX
       MOV  SI,[BX]
```

```
        MOV CX,2[BX]
        MOV DI,4[BX]
ADDPRO:ADD AL,[SI]
        ADC AH,0
        INC SI
        LOOP ADDPRO
        MOV [DI],AX
        POP DI                        ;恢复现场
        POP SI
        POP CX
        POP AX
        RET
SUM2   ENDP
CODE ENDS
    END START
```

(3) 利用堆栈传递参数,要注意断点的现场保护与恢复,参数的读取与返回一定要注意顺序,对堆栈的存储结构要非常清楚。

程序如下:

```
DATA SEGMENT
 ARRAY DB 10,20,30,40,50,60,70,80,90
 COUNT DW $ - ARRAY
 SUM   DW 0
DATA ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:DATA
MAIN   PROC   FAR
START: PUSH DS
        SUB AX,AX
        PUSH AX
        MOV AX,DATA
        MOV DS,AX
        LEA BX,ARRAY                  ;参数入栈
        PUSH BX
        LEA BX,COUNT
        PUSH BX
        LEA BX,SUM
        PUSH BX
        CALL SUM3                     ;调用子程序
        RET
MAIN   ENDP
;***********************************
;子程序名: SUM3
;功能: 求字节数组元素之和
;入口参数: 数组的起始地址、元素个数与和的偏移地址在堆栈中
;出口参数: 运算结果在 SUM 单元
;占用寄存器: AX,BX,CX,BP
;***********************************
SUM3   PROC NEAR
        PUSH AX                       ;保护现场
```

```
        PUSH BX
        PUSH CX
        PUSH BP
        MOV BP,SP
        PUSH DI
        MOV BX,[BP+12]              ;取参数 COUNT
        MOV CX,[BX]
        MOV SI,[BP+14]              ;取参数 ARRAY
        MOV DI,[BP+10]              ;取参数 SUM
        MOV AX,0
ADDPRO:ADD AL,[SI]
        ADC AH,0
        INC SI
        LOOP ADDPRO
        MOV [DI],AX
        POP DI                      ;恢复现场
        POP BP
        POP CX
        POP BX
        POP AX
        RET 6                       ;消除主程序压入的参数
SUM3    ENDP
CODE ENDS
    END START
```

为了理解堆栈传递参数的方法，本例程序在执行过程中，堆栈中内容的变化过程如图 11-2 所示。

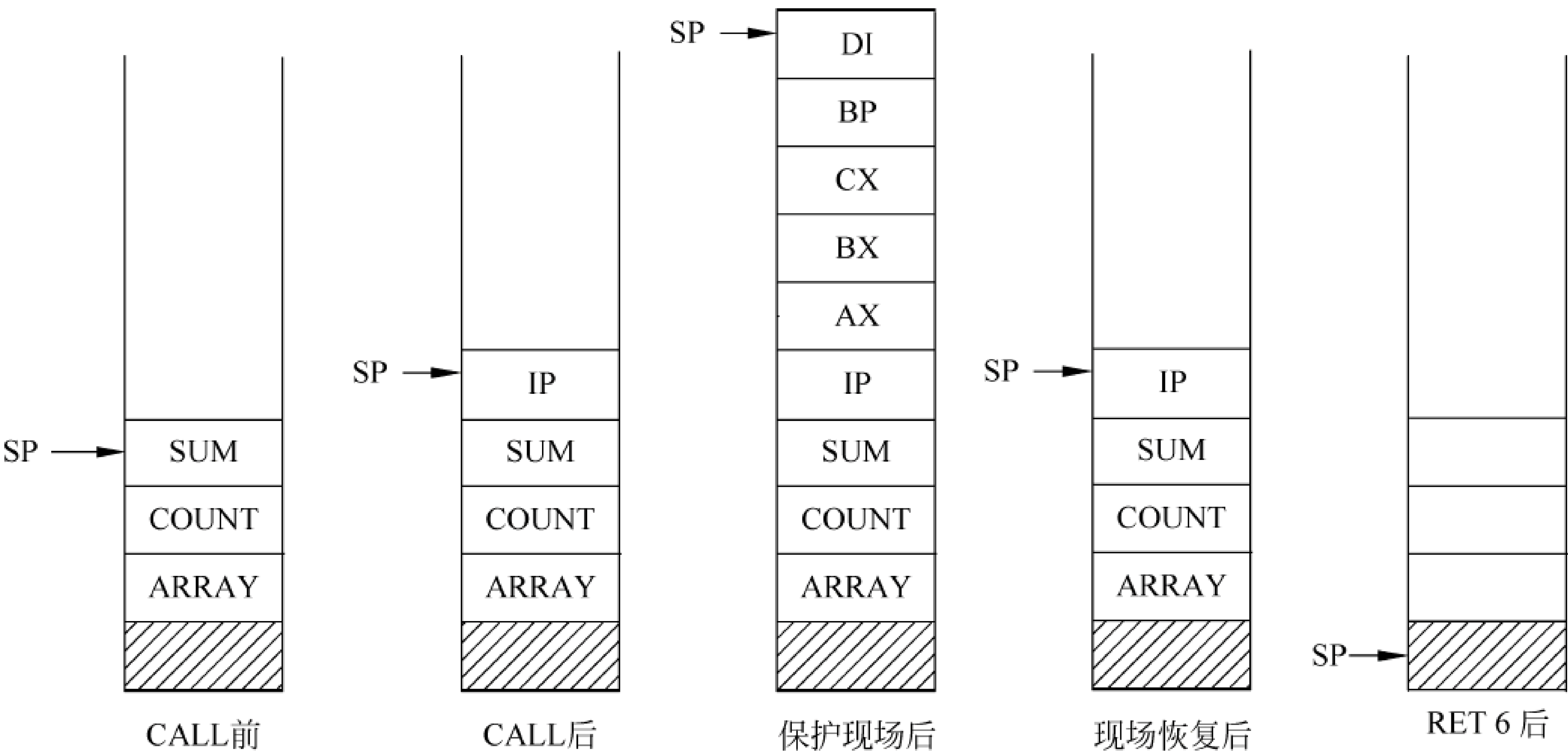


图 11-2　程序执行过程中堆栈变化情况

如果上机验证，可以在 DEBUG 中单步执行，在 CALL 命令处依然用 T 命令，这样可以跟踪到子程序内部，查看现场保护与恢复的操作情况及堆栈内容的变化。在执行 CALL SUM3 语句之前，堆栈的内容如下：

```
AX=13C5  BX=000B  CX=0053  DX=0000  SP=FFF8  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0017   NV UP EI PL ZR NA PE NC
13C6:0017 53            PUSH    BX
-T
AX=13C5  BX=000B  CX=0053  DX=0000  SP=FFF6  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0018   NV UP EI PL ZR NA PE NC
13C6:0018 E80100        CALL    001C
-D SS:FFF6
13C5:FFF0                 0B 00-09 00 00 00 00 00 B5 13        ..........
```

执行 CALL SUM3 语句之后，将子程序返回地址的 IP(001BH)压入堆栈，堆栈的内容如下：

```
-T
AX=13C5  BX=000B  CX=0053  DX=0000  SP=FFF4  BP=0000  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=001C   NV UP EI PL ZR NA PE NC
13C6:001C 50            PUSH    AX
-D SS:FFF4
13C5:FFF0           1B 00 0B 00-09 00 00 00 00 00 B5 13        ............
```

在子程序中，用于保护现场的 PUSH 语句执行后，堆栈的内容如下：

```
AX=13C5  BX=000B  CX=0053  DX=0000  SP=FFEC  BP=FFEC  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0022   NV UP EI PL ZR NA PE NC
13C6:0022 57            PUSH    DI
-T
AX=13C5  BX=000B  CX=0053  DX=0000  SP=FFEA  BP=FFEC  SI=0000  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0023   NV UP EI PL ZR NA PE NC
13C6:0023 8B5E0C        MOV     BX,[BP+0C]                SS:FFF8=0009
-D SS:FFEA
13C5:FFE0                                00 00 00 00 53 00       ....S.
13C5:FFF0  0B 00 C5 13 1B 00 0B 00-09 00 00 00 00 00 B5 13   ................
```

在子程序中，用于现场恢复的 POP 语句执行后，堆栈的内容如下：

```
-T
AX=13C5  BX=000B  CX=0053  DX=0000  SP=FFF4  BP=0000  SI=0009  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0040   NV UP EI PL NZ NA PE NC
13C6:0040 C20600        RET     0006
-D SS:FFF4
13C5:FFF0                 1B 00 0B 00-09 00 00 00 00 00 B5 13  ............
```

当执行完子程序返回语句后，堆栈的内容则回到了初始状态，内容如下：

```
AX=13C5  BX=000B  CX=0053  DX=0000  SP=FFF4  BP=0000  SI=0009  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=0040   NV UP EI PL NZ NA PE NC
13C6:0040 C20600        RET     0006
-T
AX=13C5  BX=000B  CX=0053  DX=0000  SP=FFFC  BP=0000  SI=0009  DI=0000
DS=13C5  ES=13B5  SS=13C5  CS=13C6  IP=001B   NV UP EI PL NZ NA PE NC
13C6:001B CB            RETF
```

```
- D SS:FFFC
13C5:FFF0                                             00 00 B5 13            ....
-
```

从上机验证结果看,与图 11-2 中表示的堆栈内容变化情况一致。将图示与上机验证结合起来,可以深入理解堆栈传递参数的子程序编程方法。

11.3　嵌套与递归

1. 子程序嵌套

一个主程序可以调用多个子程序,而一个子程序还可以调用另一个子程序,这称为子程序嵌套。例如,在一个主程序中遇到 CALL SUB1 指令,执行调用后则转至 SUB1 子程序;而在 SUB1 内部又有一个 CALL SUB2 指令,这时将会转到 SUB2 子程序;当执行完 SUB2 子程序后,将返回到 SUB1 中调用指令的下一条语句处继续执行;当执行完 SUB1 子程序后,将返回到主程序中调用指令的下一条语句处继续执行,完成主程序的运行,如图 11-3 所示。

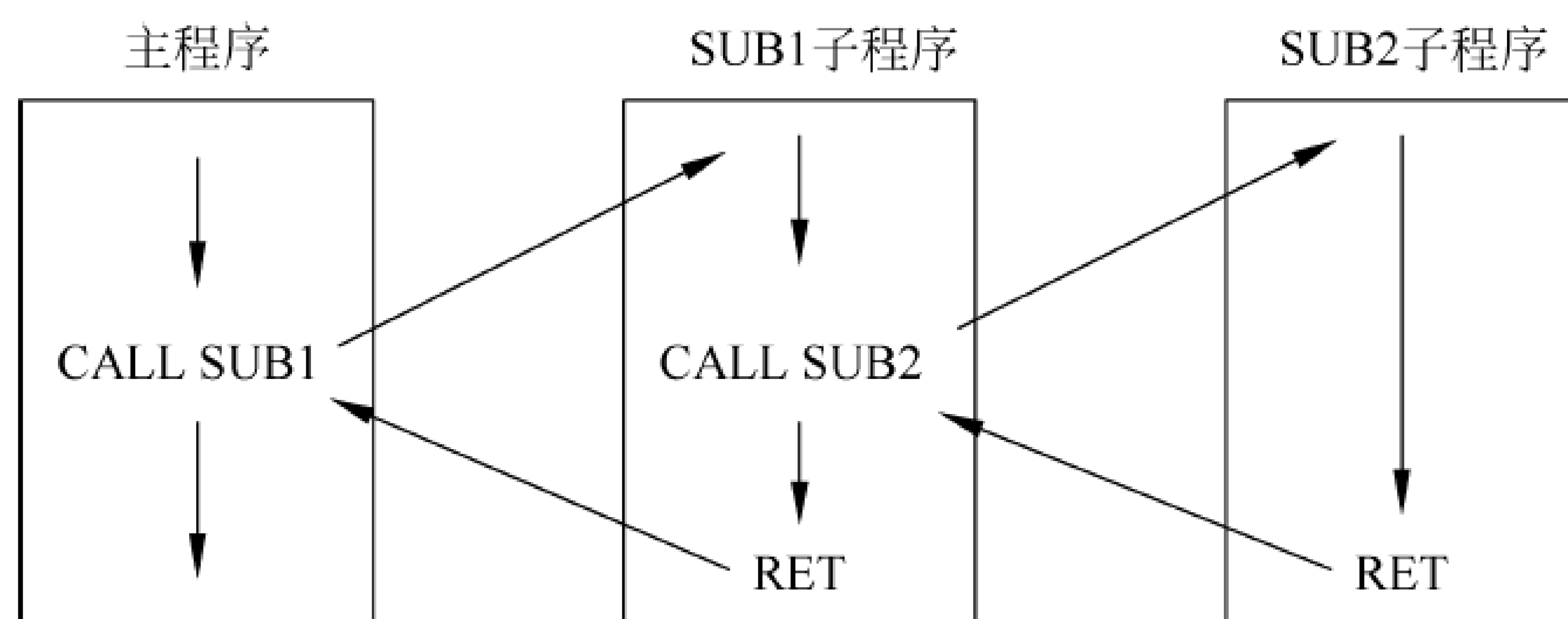


图 11-3　子程序嵌套

【例 11-4】 将一个字变量 NUM 的值转换为 4 位 ASCII 码表示的十六进制数串,串的起始地址为 STR,然后显示输出数串。

在本例中,编写子程序 SUB1,完成将一个字数据转换为 4 位 ASCII 码表示的十六进制数串,在该子程序执行过程中,调用了二级子程序 SUB2,实现将一位十六进制数转换为对应的 ASCII 码。主程序调用子程序 SUB1 实现转换操作,然后利用 2 号功能调用实现数串的显示输出。

程序如下:

```
DATA SEGMENT
  NUM   DW 2A5DH
  STR   DB 4 DUP(?)
DATA ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       LEA BX,STR
       PUSH BX                          ;将地址指针压入堆栈
```

```
        PUSH NUM                    ;将源数据压入堆栈
        CALL SUB1                   ;调用子程序 SUB1
        MOV CX,4
        LEA SI,STR
        MOV AH,2
NEXT:   MOV DL,[SI]                 ;显示输出十六进制数串
        INC SI
        INT 21H
        LOOP NEXT
        MOV AH,4CH
        INT 21H
;*********************************************
;子程序 SUB1:将一个字数据转换为四位 ASCII 码表示的十六进制数串
;入口参数:源数据地址、数串存放的起始地址 STR 压入堆栈
;出口参数:转换后的 ASCII 码在 STR 为起始地址的单元中
;*********************************************
SUB1    PROC NEAR
        PUSH BP                     ;保护现场
        MOV BP,SP
        PUSH AX
        PUSH DI
        PUSH CX
        PUSH DX
        PUSHF
        MOV AX,[BP+4]               ;取 NUM
        MOV DI,[BP+6]               ;取 STR 地址指针
        ADD DI,LENGTH STR-1         ;地址指针指向 STR 末单元
        MOV DX,AX
        MOV CX,4
AGAIN:  AND AX,0FH                  ;取出低四位
        CALL SUB2                   ;调用子程序 SUB2
        MOV [DI],AL                 ;存转换后的 ASCII 码
        PUSH CX
        MOV CL,4
        SHR DX,CL
        MOV AX,DX
        POP CX
        DEC DI
        LOOP AGAIN
        POPF
        POP DX
        POP CX
        POP DI
        POP AX
        POP BP
        RET 4
SUB1    ENDP
;**************************************************
;子程序 SUB2:将一位十六进制数转换为对应的 ASCII 码
;入口参数:AL 中低四位存放一位十六进制数
;出口参数:转换后的 ASCII 码在 AL 中
```

```
;*************************************************
SUB2    PROC NEAR
        CMP AL,9
        JBE L1
        ADD AL,7
L1:     ADD AL,30H
        RET
SUB2    ENDP
CODE    ENDS
    END START
```

2. 子程序递归

如果在子程序调用过程中,出现子程序调用其自身的情况,则称为递归调用。递归分为直接递归和间接递归两种。递归调用的关键是要有一个出口,否则程序将无法停止。

【例 11-5】 用递归方法求数的阶乘。

数 N 的阶乘定义为:$N! = N\times(N-1)\times(N-2)\times\cdots\times3\times2\times1$,如果数 N 较大,则阶乘的结果会非常大,程序会非常复杂,请读者作为课程设计练习题。这里主要为理解递归调用的思想,简化处理,限定数 N 为 1～6。

在主程序中调用子程序 FACT 求阶乘,在 FACT 子程序内部,有一个自身调用,形成递归调用。递归调用的出口条件是当寄存器 AX 的内容为 0 时返回。程序如下:

```
DATA SEGMENT
 N   DW 6
 FN DW ?
DATA ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
        MOV DS,AX
        MOV AX,N
        CALL FACT
        MOV FN,DX
        MOV AH,4CH
        INT 21H
;************************************
;子程序 FACT: 求数 N 的阶乘(1～6)
;入口参数: 数 N 在 AX 中
;出口参数: 阶乘在 DX 中
;************************************
FACT    PROC NEAR
        PUSH AX             ;保护现场
        CMP AX,0
        JA L1               ;AX 大于 0,转至 L1
        MOV DX,1            ;否则,DX←1
        JMP L2
L1:     PUSH AX             ;保存 N 的值
        DEC AX
        CALL FACT           ;调用自身
        POP AX
```

```
        MUL DL              ;N*(N-1)
        MOV DX,AX
L2:     POP AX              ;恢复现场
        RET
FACT    ENDP
CODE    ENDS
    END START
```

11.4 实验内容

【实验目的】 理解子程序的定义及其调用的概念,掌握子程序设计的一般方法,熟悉子程序的上机调试的步骤和技巧。

【实验 11-1】 编写如下子程序和主程序:

(1) 子程序 MAX:求一组字节无符号数中最大数的子程序。

(2) 子程序 OUTPUT:一个字节的十六进制数显示在屏幕上。

(3) 假设一组字节无符号数为:00H,12H,3BH,43H,60H,0CH,8AH,0ABH,37H,0FFH,32H,47H。找出最大数,并将最大数以十六进制形式显示在屏幕上。

解析:该程序采用子程序结构设计,主程序中由两个子程序组成,分别是求最大值子程序 MAX 和十六进制数显示子程序 OUTPUT。

程序如下:

```
DATA    SEGMENT
   BUFFER   DB 00H,12H,3BH,43H,60H,0CH
            DB 8AH,0ABH,37H,0FFH,32H,47H
   COUNT   EQU    $ - OFFSET BUFFER
DATA    ENDS
CODE   SEGMENT
      ASSUME   CS:CODE, DS:DATA
MAIN   PROC FAR
START:  PUSH DS
        SUB AX,AX
        PUSH AX
        MOV AX, DATA
        MOV DS, AX
        MOV BX,OFFSET BUFFER
        MOV CX,COUNT
        CALL MAX
        CALL OUTPUT
        RET
MAIN    ENDP
;**********************************************
;子程序 MAX
;功能: 求一组字节无符号数中最大数
;入口参数: 字节数组的起始地址送 BX,元素个数送 CX
;出口参数: 最大数放在 AL 中
;***********************************************
```

```
MAX     PROC NEAR
        MOV AL,[BX]
        INC BX
        DEC CX
CMPARE:CMP AL,[BX]                 ;AL 与下一个数比较
        JA NEXT                    ;若大于,则转
        MOV AL,[BX]                ;否则,把较大者送 AL
NEXT:  INC BX
        LOOP CMPARE
        RET
MAX     ENDP
;*************************************************************
;子程序 OUTPUT
;功能:一个字节的十六进制数显示在屏幕上
;入口参数:要显示的十六进制数放在 AL 中
;出口参数:无
;*************************************************************
OUTPUT PROC NEAR
        MOV  BL,AL                 ;最大数 AL 送入 BL,暂存
        MOV  DL,AL                 ;最大数 AL 送入 DL
        MOV  CL,4
        SHR  DL,CL                 ;DL 的高四位变成低四位,高四位为零
        CMP  DL,0AH                ;与 0AH 比较
        JB   L1                    ;若小于,则转
        ADD  DL, 7                 ;否则,+7
L1:     ADD  DL, 30H               ;+30H
        MOV  AH, 2
        INT  21H                   ;输出十六进制数的高位
        MOV  DL, BL                ;最大数送入 DL
        AND  DL,0FH                ;屏蔽高四位
        CMP  DL, 0AH               ;与 0AH 比较
        JB   L2
        ADD  DL, 7
L2:     ADD  DL, 30H
        MOV  AH, 2
        INT  21H                   ;输出十六进制数的低位
        RET
OUTPUT ENDP
CODE    ENDS
   END  START
```

【实验 11-2】 十进制到十六进制的转换程序。要求:从键盘输入一个十进制数,然后把该数以十六进制的形式显示在屏幕上。

解析:该程序采用子程序结构设计,主程序中由三个子程序组成,分别是十进制数字的 ASCII 码转换成二进制、二进制转换成十六进制并显示输出、回车换行。

程序如下:

```
CODE   SEGMENT
    ASSUME CS:CODE
MAIN   PROC FAR
```

```
START: PUSH DS
       SUB AX,AX
       PUSH AX
       CALL DECIBIN
       CALL NEWLINE
       CALL BINHEX
       RET
MAIN   ENDP
;*****************************************
;子程序 DECIBIN
;功能：十进制数字的 ASCII 码转换成二进制
;入口参数：无
;出口参数：转换后的二进制数在 BX 中
;*****************************************
DECIBIN  PROC NEAR
         MOV BX,0
NEWCHAR: MOV AH,1
         INT 21H
         SUB AL,30H
         JL EXIT
         CMP AL,9
         JG EXIT
         CBW
         XCHG AX,BX
         MOV CX,10
         MUL CX
         XCHG AX,BX
         ADD BX,AX
         JMP NEWCHAR
EXIT:    RET
DECIBIN  ENDP
;*****************************************
;子程序 BINHEX
;功能：二进制转换成十六进制并显示输出
;入口参数：二进制数在 BX 中
;出口参数：无
;******************************************
BINHEX PROC NEAR
         MOV CH,4
ROTATE:  MOV CL,4
         ROL BX,CL
         MOV AL,BL
         AND AL,0FH
         ADD AL,30H
         CMP AL,3AH
         JL PRINTIT
         ADD AL,7
PRINTIT: MOV DL,AL
         MOV AH,2
         INT 21H
         DEC CH
```

```
          JNZ ROTATE
          RET
BINHEX ENDP
;****************************************
;子程序 NEWLINE
;功能: 回车换行
;****************************************
NEWLINE   PROC NEAR
          MOV DL,0DH
          MOV AH,2
          INT 21H
          MOV DL,0AH
          INT 21H
          RET
NEWLINE   ENDP
CODE      ENDS
          END START
```

习　题

1. 定义两个字节变量 DA1 和 DA2,在 DA1 和 DA2 单元中各有一个无符号数,采用子程序结构的形式编程,完成两个无符号数的加法运算,并将结果以十进制形式显示在屏幕上。要求:

(1) 编写两数相加的子程序;

(2) 编写在屏幕上显示输出一个字符的子程序;

(3) 编写二进制数转换为十进制数并显示输出的子程序(假设为两位十进制数);

(4) 写出主程序调用子程序的程序段,并上机调试通过。

2. 阅读如下程序,完成:

(1) SUBP1 子程序完成什么功能?

(2) SUBP2 子程序完成什么功能?

(3) 程序执行后,运行结果是什么?

```
DATAS SEGMENT
 BUF   DW   10
 COUNT   DB   4
DATAS ENDS
CODES SEGMENT
    ASSUME CS:CODES,DS:DATAS
MAIN      PROC FAR
START:    PUSH DS
          SUB AX,AX
          PUSH AX
          MOV AX,DATAS
          MOV DS,AX
          MOV BL,COUNT
          MOV SI,BUF
```

```
L1:     PUSH SI
        CALL SUBP1
        CALL SUBP2
        POP SI
        INC SI
        DEC BL
        JNZ L1
        RET
SUBP1   PROC
        PUSH AX
        PUSH DX
L2:     MOV DL,20H
        MOV AH,2
        INT 21H
        DEC SI
        JNZ L2
        POP DX
        POP AX
        RET
SUBP1   ENDP
SUBP2   PROC
        PUSH AX
        PUSH DX
        MOV CX,4
        MOV AH,2
L3:     MOV DL,'*'
        INT 21H
        LOOP L3
        MOV DL,0DH
        INT 21H
        MOV DL,0AH
        INT 21H
        POP DX
        POP AX
        RET
SUBP2   ENDP
MAIN   ENDP
CODES ENDS
    END START
```

第 12 章

中断与 DOS 功能调用

在传统的磁盘操作系统(MS-DOS)中,有 BIOS 层的功能模块和 DOS 层的功能模块可以供用户调用,实现输入、输出、存储管理、打印机管理、文件管理等基本操作。本章主要介绍中断以及常见的 DOS 功能调用方法。

12.1 中　　断

12.1.1 中断及中断处理

中断是指程序在运行过程中出现了某种紧急事件,必须终止现行程序而转去执行一个处理该事件的程序,处理结束后又返回到原来被暂停的程序继续执行。能引起中断的事件称为中断源,常见的中断源分为硬件中断和软件中断。硬件中断是由硬件产生的中断信号,如硬件故障、掉电等。软件中断是由特定的指令引起的,如 INT 指令。CPU 执行 INT 指令,会立即产生中断,去调用系统中相应的中断服务程序来完成中断功能。

中断服务程序是处理紧急事件的专用程序,由三部分组成:保护原程序的断点和寄存器等现场信息;处理该事件;恢复断点和现场信息。中断服务程序的起始地址称为中断入口。

把中断服务程序的入口地址放在一起建立的表称为中断向量表,占内存单元的地址为 0000:0000～0000:03FFH;每四个单元内容构成一个中断入口地址(CS:IP);表中内容分为 256 项,对应于类型号 0～255,如图 12-1 所示。



图 12-1　中断向量表

12.1.2 中断向量的设置

在中断向量表中，类型号小的级别高，而类型号大的级别低。表 12-1 列出了中断类型号及其对应的地址分配。

表 12-1 中断向量的地址分配

地址(十六进制)	类型号(十六进制)	中 断 作 用
0～1F	0～7	系统保留
20～3F	8～F	8259 中断控制器芯片专用
40～7F	10～1F	BIOS 中断
80～9F	20～27	DOS 中断
A0～FF	28～3F	DOS 保留
100～17F	40～5F	扩充 BIOS 中断向量
180～19F	60～67	用户中断向量
1A0～1BF	68～6F	保留
1C0～1DF	70～77	I/O 中断向量
1E0～1FF	78～7F	保留
200～217	80～85	BASIC 保留
218～3C3	86～F0	BASIC 中断向量
3C4～3FF	F1～FF	保留

当有多个中断源时，CPU 根据中断优先权进行排队，然后按优先级别从高到低的次序来依次处理各个中断源的中断请求。响应多个中断的原则：高级别的中断可以打断级别比它低的中断，产生中断嵌套；低级别的中断不能打断级别比它高的中断，也不能被同级别的中断所打断。

12.1.3 DOS 中断

从表 12-1 中可以看出，类型号 20H～27H 为 DOS 所使用的中断，用户可以通过 INT 20H～INT 27H 进入相应的 DOS 中断服务程序。

1. 20H 类中断

程序退出的传统方法是使用 INT 20H，使程序正常退出。在 DEBUG 环境下，可以使用这个中断指令结束程序，但在汇编语言程序中一般建议使用 4CH 功能调用，来结束程序返回 DOS 提示符下。

2. 21H 类中断

这类中断是 DOS 内部功能调用，将在下一节中详细介绍，这是本章的重点内容。

3. 22H～27H 类中断

这些类型的中断可参阅有关专业技术手册，在此不予介绍。

12.2 DOS 功能调用

在汇编语言层次编写输入输出或文件处理等应用程序时，使用者必须各自提供所需的基本处理。在 DOS 系统中，将一些基本处理当作子程序建于 DOS 系统内，让使用者可以调

用,这就是 DOS 功能调用。

DOS 提供了大量的各种子程序功能调用,涉及字符 I/O、文件管理、内存管理、日期和时间设置、查询功能及其他多种功能。当启动计算机后,DOS 系统功能调用的子程序就驻入内存,例如前面章节已经用过的键盘输入一个字符和字符显示输出都是这类功能调用。

12.2.1　调用方法

在 DOS 中断指令中,INT 21H 是进入各功能调用子程序的总入口。每个功能调用规定一个功能号,这个功能号就相当于进入 DOS 系统功能调用中相应的各个子程序入口,不同的功能号对应于不同的处理功能。

DOS 系统功能调用的使用方法:

```
一般格式:MOV    AH,功能号
         [设置入口参数]
         INT    21H
         [设置出口参数]
```

12.2.2　常见的几种功能调用

1. 1 号 DOS 功能调用

功能:键盘输入一个字符,有回显。

入口参数:AH←01H。

出口参数:AL←所输入的字符 ASCII。

例如:

```
MOV   AH,01H
INT   21H
```

执行 1 号 DOS 功能调用时,系统等待用户从键盘输入一个字符,将该字符的 ASCII 码存入 AL 寄存器,同时把输入的字符回送到屏幕上显示。

2. 2 号 DOS 功能调用

功能:显示输出一个字符。

入口参数:AH←02H;DL←待输出的字符的 ASCII 码。

出口参数:无。

例如:

```
MOV   DL,'5'
MOV   AH,02H
INT   21H
```

执行 2 号 DOS 功能调用时,在屏幕上显示 DL 中存放的字符。

3. 5 号功能调用

功能:打印输出一个字符。

入口参数:AH←05H;DL←待输出的字符的 ASCII 码。

出口参数:无。

例如：

```
MOV  DL,'5'
MOV  AH,05H
INT  21H
```

执行 5 号 DOS 功能调用时，在打印机上输出 DL 中存放的字符。

4. 7 号 DOS 功能调用

功能：键盘输入一个字符，无回显。

入口参数：AH←07H。

出口参数：AL←所输入的字符 ASCII。

执行 7 号 DOS 功能调用时，系统等待用户从键盘输入一个字符，将该字符的 ASCII 码存入 AL 寄存器。

5. 9 号 DOS 功能调用

功能：显示输出字符串。

入口参数：AH←09H；DS:DX←字符串的起始地址。

出口参数：无。

说明：字符串必须以'$'结尾。

6. 0AH 号 DOS 功能调用

功能：缓冲区键盘输入。

入口参数：AH←0AH；DS:DX ←输入缓冲区的起始地址。

出口参数：无。

说明：输入字符串从缓冲区起始地址+2 处开始存储。

0AH 号 DOS 功能调用不同于 1 号 DOS 功能调用，它能接收键盘输入的字符串，存入到指定的缓冲区。它要求缓冲区的第一个字节单元存放允许输入的最大字符数，第二个字节单元存放实际输入的字符数，从第三个字节单元才开始存放输入的字符串。输入缓冲区结构如图 12-2 所示。

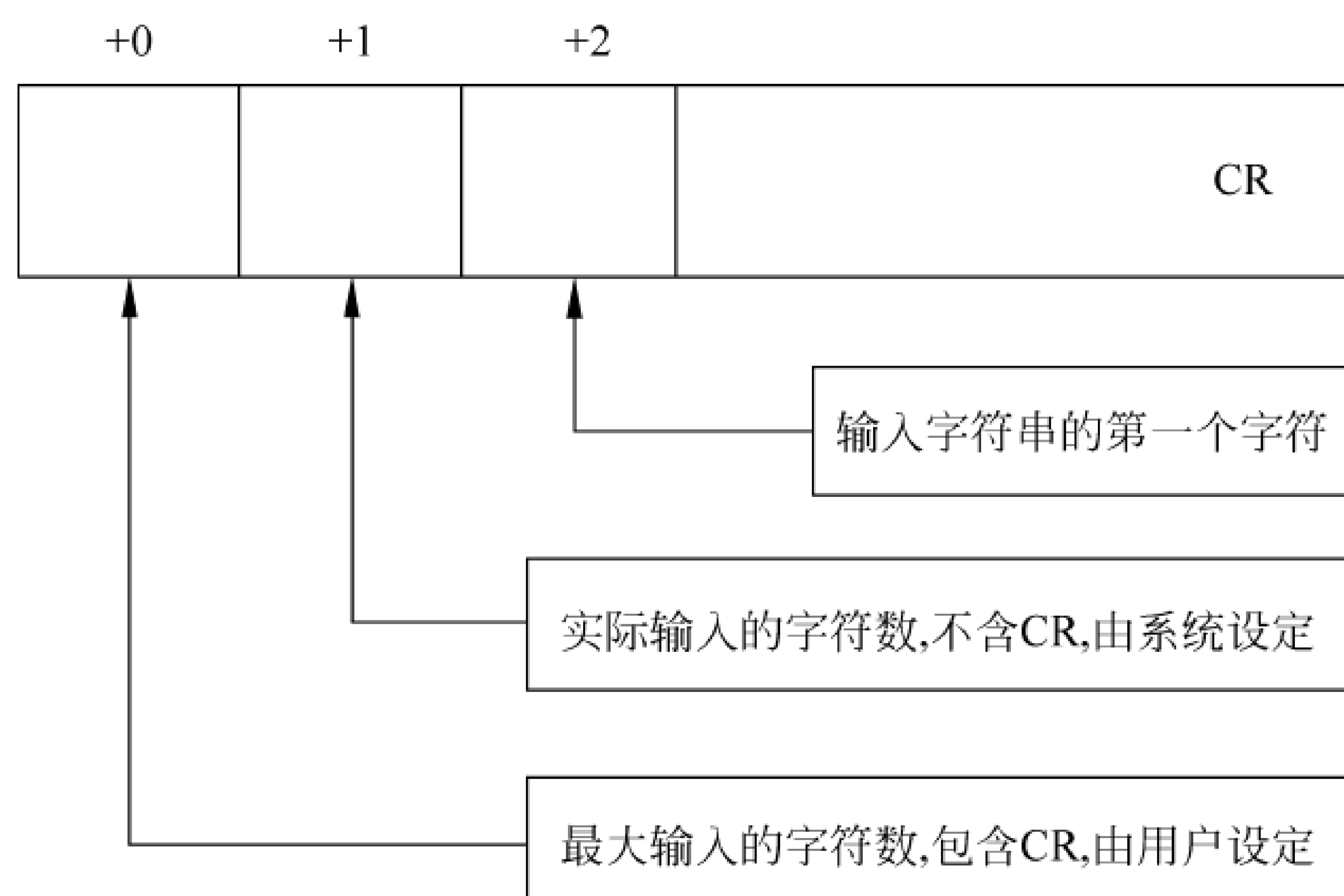


图 12-2　输入缓冲区结构示意图

12.2.3　DOS 功能调用应用举例

【例 12-1】 先显示输出一行信息，然后根据输入的 Y/N 的不同，分别显示不同的信息。

在定义字符串时，每个字符串都是以'$'结尾，这是采用 9 号 DOS 功能调用所要求的。如果采用 2 号 DOS 功能调用，定义字符串时就不需要以'$'结尾。2 号 DOS 功能调用每次输出一个字符，如果输出字符串，需要利用循环结构实现，程序结构复杂，有兴趣的读者可以自行完成采用 2 号 DOS 功能调用的编程。

采用 9 号 DOS 功能调用实现的程序：

```
DATA       SEGMENT
  TP       DB 'Is it after 12 noon (Y/N)? $ '
  AM       DB 0DH,0AH,'Good morning,friends! $ '
  PM       DB 0DH,0AH,'Good afternoon,world! $ '
DATA       ENDS
CODE       SEGMENT
   ASSUME CS:CODE,DS:DATA
START:     MOV AX,DATA
           MOV DS,AX
           MOV AH,09H
           MOV DX,OFFSET TP
           INT 21H                ;输出字符串 TP
           MOV AH,01H
           INT 21H                ;接收键盘输入
           CMP AL,'Y'
           JE IsAfter
           CMP AL,'y'
           JE IsAfter
           MOV AH,09H
           MOV DX,OFFSET AM
           INT 21H                ;输出字符串 AM
           JMP END_L
IsAfter:   MOV AH,09H
           MOV DX,OFFSET PM
           INT 21H                ;输出字符串 PM
END_L:     MOV AH,4CH
           INT 21H
CODE       ENDS
           END START
```

在集成实验环境下运行的效果如图 12-3 所示。程序中在键盘输入大写'Y'或小写'y'均可，但输入字符的有效性并没有检测。如果要完善该程序，需要增加输入字符的有效性检查，读者可以自己完成。



图 12-3　根据 Y/N 的不同显示输出

【例 12-2】 程序运行后,先显示信息"Who are you ?",等待用户输入自己的名字,之后再将用户输入的字符串重新显示一行。

在程序中,采用了 9 号 DOS 功能调用输出"Who are you ?",然后利用 0AH 号 DOS 功能调用接收用户输入的字符串。由于接收的字符串末尾没有'$'字符,所以本程序中采用了 2 号 DOS 功能调用,用循环结构实现字符串的输出。程序如下:

```
DATA   SEGMENT
  HRU       DB 'Who are you? $ '
  MyName    DB 20,?,20 DUP(?)
DATA    ENDS
CODE    SEGMENT
   ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       MOV AH,09H
       MOV DX,OFFSET HRU
       INT 21H                    ;输出字符串 HRU
       MOV AH,0AH
       MOV DX,OFFSET MyName
       INT 21H                    ;接收用户输入字符串
       MOV AH,02H
       MOV DL,0DH
       INT 21H
       MOV DL,0AH
       INT 21H                    ;回车换行
DISP:  MOV CL,MyName[1]           ;CL←实际输入的字符数
       MOV CH,0
       MOV BX,OFFSET MyName[2]    ;字符串存储的起始地址
NEXT:  MOV AH,02H
       MOV DL,[BX]
       INT 21H                    ;输出一个字符
       INC BX
       LOOP NEXT
       MOV AH,4CH
       INT 21H
CODE    ENDS
       END START
```

如果要采用 9 号 DOS 功能调用,还必须在用户输入的字符串后添加'$'字符,这就要求对输入缓冲区的结构理解透彻。例 12-2 改进后的程序如下:

```
DATA   SEGMENT
  HRU       DB 'Who are you? $ '
  MyName    DB 20,?,20 DUP(?)
DATA    ENDS
CODE    SEGMENT
   ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       MOV AH,09H
```

```
        MOV DX,OFFSET HRU
        INT 21H                     ;输出字符串 HRU
        MOV AH,0AH
        MOV DX,OFFSET MyName
        INT 21H                     ;接收用户输入字符串
        MOV AH,02H
        MOV DL,0DH
        INT 21H
        MOV DL,0AH
        INT 21H                     ;回车换行
        MOV BL,MyName[1]            ;AL←实际输入的字符数
        ADD BL,2
        MOV BH,0
        MOV MyName[BX],'$'          ;字符串末尾加'$'
        MOV DX,OFFSET MyName[2]     ;字符串的起始地址
DISP:   MOV AH,09H
        INT 21H                     ;输出字符串
        MOV AH,4CH
        INT 21H
CODE    ENDS
        END START
```

在集成实验环境下运行的效果如图 12-4 所示。程序运行时，先显示输出信息“Who are you ?”，用户输入“James”，然后再重新显示一行“James”。

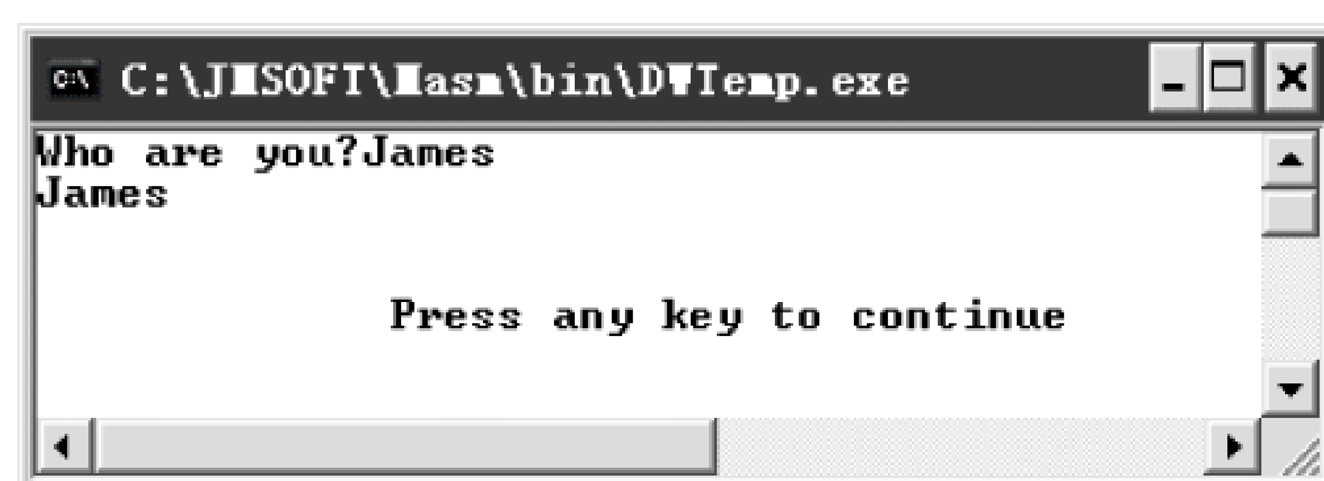


图 12-4　例 12-2 程序运行效果

12.3　实 验 内 容

【实验目的】 理解常用的 DOS 功能调用的基本功能，能够熟练运用 1 号、2 号、9 号、10 号 DOS 功能调用编写字符的输入输出程序，掌握上机调试方法。

【实验 12-1】 在键盘上输入 10 个字符存入 S1 开始的存储单元，然后以与输入字符的先后相同的顺序显示出来。如果是逆序输出，应如何修改程序？请上机调试程序，查看运行结果。

```
;采用 1 号 DOS 功能调用和 2 号 DOS 功能调用，实现与输入顺序相同正序输出
DATA SEGMENT
    S1  DB 10 DUP(?)
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
```

```
        MOV DS,AX
        MOV  SI,0
        MOV  CX,10
INPUT: MOV  AH,1
        INT 21H                 ;输入一个字符
        MOV S1[SI],AL
        INC  SI
        LOOP INPUT
        MOV AH,02H
        MOV DL,0DH
        INT 21H
        MOV DL,0AH
        INT 21H                 ;回车换行
        MOV CX,10
        MOV DI,0
DISP:   MOV DL,S1[DI]
        MOV AH,2
        INT 21H                 ;输出一个字符
        INC DI
        LOOP DISP
        MOV AH,4CH
        INT 21H
CODE ENDS
    END START
;采用 0AH 号 DOS 功能和 9 号 DOS 功能调用,实现与输入顺序相同正序输出
DATA SEGMENT
    S1  DB 11,?,11 DUP(?)
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
        MOV DS,AX
INPUT: MOV DX,OFFSET S1
        MOV AH,0AH
        INT 21H                 ;输入 10 个字符
        MOV BX,OFFSET S1
        MOV AL,S1 + 1           ;输入的字符数送 AL
        MOV AH,0
        ADD BX,AX
        MOV 2[BX],'$'           ;末尾添加'$'
        MOV AH,02H
        MOV DL,0DH
        INT 21H
        MOV DL,0AH
        INT 21H                 ;回车换行
DISP:   LEA DX,S1 + 2           ;待显示字符串的起始地址
        MOV AH,9
        INT 21H                 ;输出一个字符
        MOV AH,4CH
        INT 21H
CODE ENDS
```

```
    END START
;采用 1 号 DOS 功能和 2 号 DOS 功能调用,实现逆序输出
DATA SEGMENT
    S1  DB 10 DUP(?)
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       MOV  SI,0
       MOV  CX,10
INPUT: MOV  AH,1
       INT 21H                 ;输入一个字符
       MOV S1[SI],AL
       INC  SI
       LOOP INPUT
       MOV AH,02H
       MOV DL,0DH
       INT 21H
       MOV DL,0AH
       INT 21H                 ;回车换行
       MOV CX,10
       MOV DI,9                ;DI 初值为末单元的偏移量
DISP:  MOV DL,S1[DI]
       MOV AH,2
       INT 21H                 ;输出一个字符
       DEC DI                  ;逆序指向下一个单元
       LOOP DISP
       MOV AH,4CH
       INT 21H
CODE ENDS
    END START
```

【实验 12-2】 程序运行时,先显示信息“Pls input N:”,然后从键盘上输入一个 3~9 的数字 N,则输出一个 N 行 N 列 * 号组成的方块。要求检查输入的有效性,对于无效数据提示重新输入。例如输入 3,则输出:

```
***
***
***
```

参考程序如下:

```
DATA   SEGMENT
Message   DB 'Pls input N: $ '
Err_Msg   DB 'Invalid character,input again. N: $ '
DATA   ENDS
CODE   SEGMENT
    ASSUME CS:CODE,DS:DATA
START:   MOV AX,DATA
         MOV DS,AX
```

```
        MOV AH,09H
        MOV DX,OFFSET Message
        INT 21H                     ;显示信息
        JMP AGAIN
ErrMsg: MOV AH,09H
        MOV DX,OFFSET Err_Msg
        INT 21H                     ;显示错误信息
AGAIN:  MOV AH,1
        INT 21H                     ;输入一个数字字符
        CMP AL,'3'
        JB  ErrMsg
        CMP AL,'9'
        JA  ErrMsg
        SUB AL,30H                  ;ASCII to BCD
        MOV CL,AL
        MOV CH,0
        MOV AH,2
        MOV DL,0DH
        INT 21H
        MOV DL,0AH
        INT 21H                     ;回车换行
        MOV BX,CX
L2:     PUSH CX
        MOV CX,BX
L1:     MOV DL,'*'
        MOV AH,02H
        INT 21H                     ;输出*
        LOOP L1
        MOV DL,0DH
        INT 21H
        MOV DL,0AH
        INT 21H                     ;回车换行
        POP CX
        LOOP L2
        MOV AH,4CH
        INT 21H
CODE   ENDS
    END START
```

习　　题

1. 从 STR1 为起始地址的 30 个字符,依次传送到以 STR2 为起始地址的连续字节存储单元中。设 STR1 和 STR2 两个存储区域不发生重叠。分别完成:

(1) 按正序将 STR2 中的字符依次显示输出。

(2) 按逆序将 STR2 中的字符依次显示输出。

2. 从键盘输入一个字符串,然后统计出数字字符的个数、英文字母的个数及其他字符的个数,并将统计结果显示输出。

3. 从键盘输入一个字符串，将其中的小写字母转换为大写字母、大写字母转换为小写字母、数字不转换、其他字符转换为'#'，把转换后的字符串显示输出。

4. 从键盘上输入两个一位数，相加后以十进制形式输出和。要求：运行程序时，出现提示信息"Please input expression:"，输入第一个数字后，显示"+"号后，再输入第二个数，然后显示"="号和结果。例如，"Please input expression: 6+9=15"。

5. 阅读如下程序，完成：

(1) 在该程序中，用到了哪些 DOS 功能调用？

(2) STRCMP 子程序完成什么功能？

(3) 该程序完成什么功能？

```
DATAS SEGMENT
   BUF1  DB  'ABCDEFGHIJ'
   BUF2  DB  11,0,11 DUP(0)
   EQL   DB  'STRING EQUAL! $ '
   NEQ   DB   'STRING NON-EQUAL!$ '
   FLAG  DB  0
DATAS ENDS
CODES SEGMENT
    ASSUME CS:CODES,DS:DATAS
MAIN  PROC FAR
START:PUSH DS
      SUB AX,AX
      PUSH AX
      MOV AX,DATAS
      MOV DS,AX
      MOV ES,AX
      LEA DX,BUF2
      MOV AH,0AH
      INT 21H
      MOV CX,10
      LEA SI,BUF1
      LEA DI,BUF2 + 2
      CALL STRCMP
      CMP FLAG,0
      JZ NEXT
      LEA DX,EQL
      MOV AH,9
      INT 21H
      JMP EXIT
NEXT: LEA DX,NEQ
      MOV AH,9
      INT 21H
EXIT: RET
MAIN  ENDP
STRCMP  PROC
      PUSH AX
      CLD
      REPE CMPSB
```

```
        JNZ R
        MOV FLAG, - 1
        JMP R1
R:      MOV FLAG,0
R1:     POP AX
        RET
STRCMP  ENDP
CODES ENDS
      END START
```

第13章

宏汇编技术

宏与子程序类似，可以作为一个独立的功能程序被其他程序多次调用，使用宏功能可增强汇编语言程序的可读性，简化程序结构。本章主要介绍宏功能的使用过程，了解重复汇编和条件汇编的使用。

13.1 宏 汇 编

使用子程序结构的程序设计，虽然具有很多优点，但是要增加额外的存储空间和程序运行时间的开销。因此，在子程序本身较短或者对占用存储空间及运行速度没有特殊要求，或者是需要传递的参数较多的情况下，使用宏汇编就更加有利而且灵活。

如果在汇编语言程序中，有的程序段在整个程序中要多次出现，这种出现有的可能是完全不修改的重复，有的可能是仅修改程序段中某些操作数字段，而程序段的功能并没有多大的变化。为了减少在程序中重复编写相同程序段的工作，可以采用宏定义方式编写程序段，供程序进行宏调用，在汇编时再进行宏展开。

宏功能的使用过程分为：宏定义、宏调用、宏展开。必须掌握“先定义，后调用”的基本原则。宏功能的使用，可以使汇编语言程序更加清晰，易于阅读，简化重复程序段的编写，减少重复编写相同程序段时出错的可能性，使用起来更加灵活。

13.1.1 宏定义

用 MACRO/ENDM 伪指令进行宏定义有以下两种格式。

(1) 不带参数的宏定义：

```
<宏名>    MACRO
            ⋮ }宏体
            ENDM
```

(2) 带参数的宏定义：

```
<宏名>    MACRO    形参 1,形参 2,…
            ⋮ }宏体
            ENDM
```

其中，宏体是由若干条指令语句所组成的程序段。宏名就是给该宏体中的程序段指定的一个符号名。注意：在 ENDM 语句之前不写宏名。

不带参数的宏定义：每次宏调用时，宏体内各语句序列均不做任何修改。

【例 13-1】 使用宏定义实现回车换行的操作。

```
NEWLINE  MACRO
         MOV AH,2
         MOV DL,0DH
         INT 21H
         MOV DL,0AH
         INT 21H
         ENDM
```

带参数的宏定义：宏定义用到的每个形式参数之间用逗号分隔。有了形式参数，使得在宏体中程序段的某些部分，允许在宏调用时做适当的修改。在宏定义中，把允许修改的部分用形式参数表示。当宏调用时，就用相应的实际参数来替代。

【例 13-2】 使用宏定义实现对两个字节单元内容互换的操作。

```
SWAP  MACRO  X,Y
      PUSH   AX            ;保护
      MOV    AL,X
      XCHG   AL,Y
      MOV    X,AL
      POP    AX            ;恢复
      ENDM
```

其中，X 和 Y 是可以临时指定进行互相交换数据的两个字节存储单元的变量。在宏定义中 X、Y 为形参。

13.1.2 宏调用

经过宏定义之后，在程序中使用宏时，可以直接引用宏名，构成宏指令语句，这个过程称为宏调用。

宏调用的格式：

(1) 无参数宏调用：宏名

(2) 带参数宏调用：宏名　实参 1,实参 2,…

宏汇编程序(如 MASM)对带参数的宏调用，是用第一个实参替代第一个形参，第二个实参替代第二个形参，依次类推。一般来说，实参的个数应该和形参的个数相等，但汇编程序并不要求它们必须相等。若实参个数多于形参个数，则多余的实参不予考虑；若实参个数少于形参个数，则多余的形参做“空”处理。

【例 13-3】 利用宏功能，实现 BLOCK1 和 BLOCK2 两个字节单元内容互换。

程序如下：

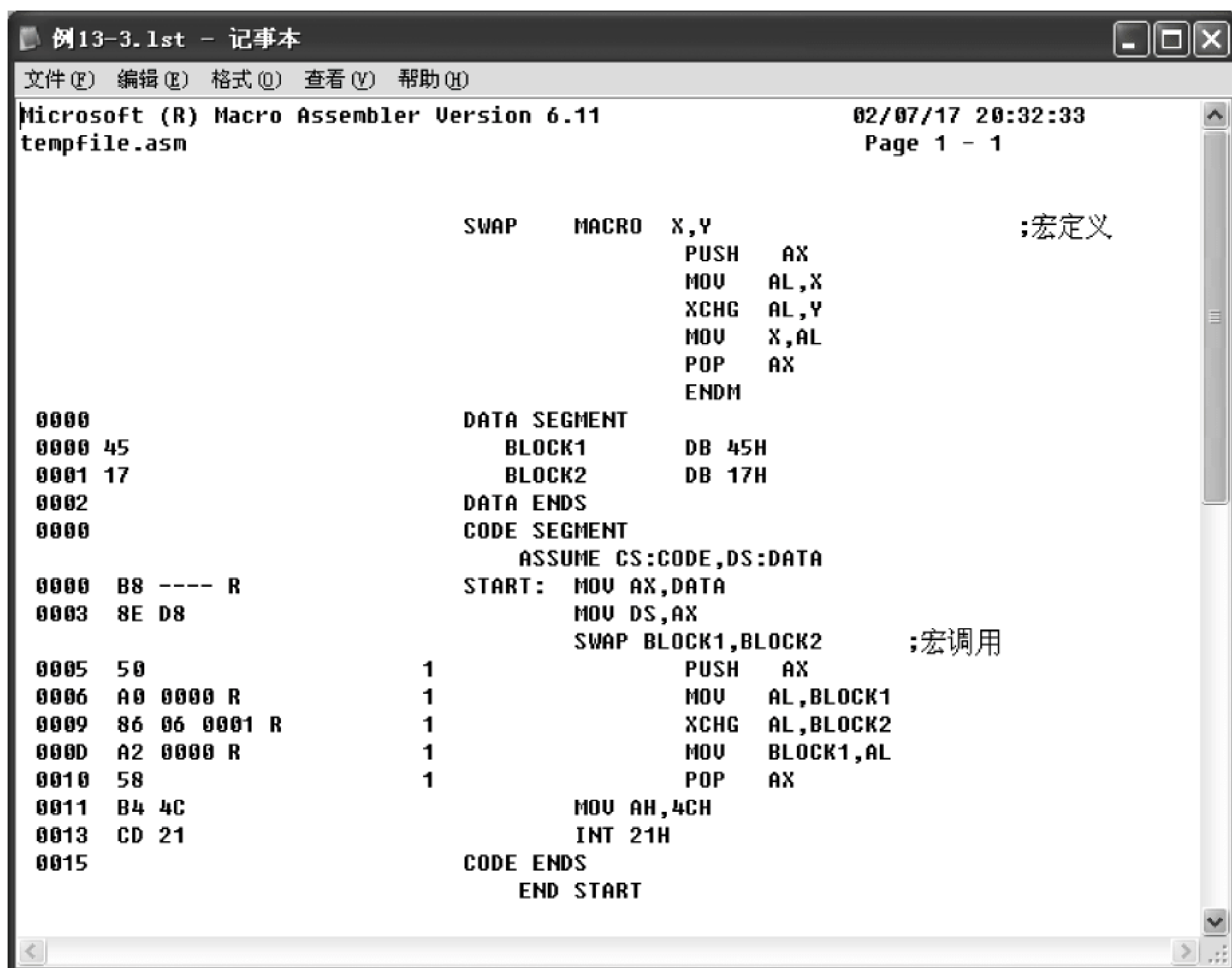
```
SWAP  MACRO  X,Y                ;宏定义
      PUSH   AX
      MOV    AL,X
      XCHG   AL,Y
      MOV    X,AL
      POP    AX
      ENDM
DATA SEGMENT
```

```
    BLOCK1   DB 45H
    BLOCK2   DB 17H
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:MOV AX,DATA
        MOV DS,AX
        SWAP BLOCK1,BLOCK2            ;宏调用
        MOV AH,4CH
        INT 21H
CODE ENDS
    END START
```

13.1.3　宏展开

在汇编过程中，当宏汇编程序扫描到宏调用的宏指令语句时，就用宏定义中宏体的目标代码替换宏指令语句，这个过程称为宏展开。若是带参数的宏调用，则同时用相应的实参替代对应的形参，并对原有宏体代码做修改。宏展开由宏汇编程序自动完成，无须用户干预。

要查看宏展开情况，可以用记事本打开汇编过程中生成的.LST 列表文件，例 13-3 中程序生成的列表文件如图 13-1 所示。可以看出，原来的宏指令处已经被若干条汇编指令所替代，形参被实参取代。


例13-3.lst - 记事本
文件(F)　编辑(E)　格式(O)　查看(V)　帮助(H)

```
Microsoft (R) Macro Assembler Version 6.11              02/07/17 20:32:33
tempfile.asm                                             Page 1 - 1

                                SWAP    MACRO  X,Y                    ;宏定义
                                                PUSH   AX
                                                MOV    AL,X
                                                XCHG   AL,Y
                                                MOV    X,AL
                                                POP    AX
                                                ENDM
 0000                           DATA SEGMENT
 0000 45                           BLOCK1       DB 45H
 0001 17                           BLOCK2       DB 17H
 0002                           DATA ENDS
 0000                           CODE SEGMENT
                                    ASSUME CS:CODE,DS:DATA
 0000  B8 ---- R                START:  MOV AX,DATA
 0003  8E D8                            MOV DS,AX
                                        SWAP BLOCK1,BLOCK2       ;宏调用
 0005  50                    1                  PUSH   AX
 0006  A0 0000 R             1                  MOV    AL,BLOCK1
 0009  86 06 0001 R          1                  XCHG   AL,BLOCK2
 000D  A2 0000 R             1                  MOV    BLOCK1,AL
 0010  58                    1                  POP    AX
 0011  B4 4C                            MOV AH,4CH
 0013  CD 21                            INT 21H
 0015                           CODE ENDS
                                    END START
```

图 13-1　例 13-3 中列表文件显示的结果

宏调用的工作方式和子程序调用的工作方式不同，子程序是在程序执行期间由主程序调用的，它只占有它自身大小的一个空间；而宏调用则是在汇编期间展开的，每调用一次就

把宏定义体展开一次,因而它占有空间与调用次数有关。一般来说,代码较长的功能段往往使用子程序;而代码较短且变元较多的功能段,使用宏功能较好。

13.1.4 LOCAL 伪操作

在一个宏定义中,如果存在变量名或标号在同一程序中多次进行宏调用,宏展开时将会产生多个相同的变量名或标号,产生重复定义符号的汇编错误。为了解决这个问题,可以使用 LOCAL 伪指令。

格式:**LOCAL** 局部标号列表

功能:汇编程序对局部标号列表中的每一个标号生成一个特殊的符号(范围??0000~??FFFF)代替宏展开中存在的每一个局部标号,以避免符号重复定义的发生。

使用这个伪指令时,要求 LOCAL 只能在宏体内,必须是 MACRO 伪指令后的第一个语句,MACRO 和 LOCAL 之间不允许有注释和分号标志,局部标号列表内的各标号之间用逗号间隔。

【例 13-4】 定义宏 HEXASC,将一位十六进制数转换成 ASCII 码,然后显示输出。

```
HEXASC   MACRO
      LOCAL L1
      CMP   DL,0AH
      JB    L1
      ADD   DL,7
L1:   ADD   DL,30H
      MOV   AH, 2
      INT   21H
      ENDM
```

下面的程序是将四位十六进制数显示输出,程序中多次调用了宏 HEXASC。程序如下:

```
DATA    SEGMENT
  BUFFER   DW 1A2BH
DATA    ENDS
CODE   SEGMENT
   ASSUME   CS:CODE, DS:DATA
START: MOV AX, DATA
       MOV DS, AX
       MOV SI,OFFSET BUFFER
       MOV DL,[SI + 1]              ;低位数送 DL
       MOV   BL,DL                  ;送入 BL 暂存
       MOV   CL,4
       SHR   DL,CL                  ;将 DL 的高四位变成低四位
       HEXASC                       ;输出十六进制数的高位
       MOV   DL, BL                 ;低位数送入 DL
       AND   DL,0FH                 ;屏蔽高四位
       HEXASC
       MOV   DL,[SI]                ;高位数送 DL
       MOV   BL,DL                  ;送入 BL 暂存
       MOV   CL,4
```

```
        SHR   DL,CL                    ;将 DL 的高四位变成低四位
        HEXASC                         ;输出十六进制数的高位
        MOV   DL, BL                   ;高位数送入 DL
        AND   DL,0FH                   ;屏蔽高四位
        HEXASC
        MOV   AH,4CH
        INT   21H
CODE    ENDS
    END  START
```

经过汇编后，从生成的.LST 列表文件可以看出，宏展开时宏体内的标号 L1，分别用符号??0000、??0001、??0002、??0003 替代。列表文件部分内容如下：

```
0000                    DATA   SEGMENT
0000 1A2B                BUFFER  DW 1A2BH
0002                    DATA   ENDS
0000                    CODE  SEGMENT
                        ASSUME  CS:CODE, DS:DATA
0000   B8 ---- R            START:  MOV   AX, DATA
0003   8E D8                        MOV   DS, AX
0005   BE 0000 R                    MOV   SI,OFFSET BUFFER
0008   8A 54 01                     MOV   DL,[SI+1]            ;低位数送 DL
000B   8A DA                        MOV   BL,DL                ;送入 BL 暂存
000D   B1 04                        MOV   CL,4
000F   D2 EA                        SHR   DL,CL                ;将 DL 的高四位变成低四位
                                    HEXASC                     ;输出十六进制数的高位
0011   80 FA 0A          1          CMP   DL,0AH
0014   72 03             1          JB   ??0000
0016   80 C2 07          1          ADD DL,7
0019   80 C2 30          1  ??0000: ADD   DL,30H
001C   B4 02             1          MOV   AH, 2
001E   CD 21             1          INT   21H
0020   8A D3                        MOV   DL, BL              ;低位数送入 DL
0022   80 E2 0F                     AND   DL,0FH              ;屏蔽高四位
                                    HEXASC
0025   80 FA 0A          1          CMP   DL,0AH
0028   72 03             1          JB    ??0001
002A   80 C2 07          1          ADD   DL,7
002D   80 C2 30          1  ??0001: ADD   DL,30H
0030   B4 02             1          MOV   AH, 2
0032   CD 21             1          INT   21H
0034   8A 14                        MOV   DL,[SI]              ;高位数送 DL
0036   8A DA                        MOV   BL,DL                ;送入 BL 暂存
0038   B1 04                        MOV   CL,4
003A   D2 EA                        SHR   DL,CL                ;将 DL 的高四位变成低四位
                                    HEXASC                     ;输出十六进制数的高位
003C   80 FA 0A          1          CMP   DL,0AH
003F   72 03             1          JB    ??0002
0041   80 C2 07          1          ADD   DL,7
0044   80 C2 30          1  ??0002: ADD   DL,30H
0047   B4 02             1          MOV   AH, 2
```

```
0049   CD 21            1           INT   21H
004B   8A D3                        MOV   DL, BL                 ;高位数送入 DL
004D   80 E2 0F                     AND   DL,0FH                 ;屏蔽高四位
                                    HEXASC
0050   80 FA 0A         1           CMP   DL,0AH
0053   72 03            1           JB    ??0003
0055   80 C2 07         1           ADD   DL,7
0058   80 C2 30         1   ??0003: ADD   DL,30H
005B   B4 02            1           MOV   AH, 2
005D   CD 21            1           INT   21H
005F   B4 4C                        MOV   AH,4CH
0061   CD 21                        INT   21H
0063                        CODE   ENDS
                                END   START
```

13.1.5 宏库及其使用

对于经常使用的宏命令，可以将它们集中存放在一个文件中，该文件称为宏库，一般用扩展名 MAC 来表示。当应用程序中需要使用宏库中的宏定义时，需要先用伪指令 INCLUDE 将宏库加入到自己的程序文件中，这样就可以按照宏定义的要求调用。

格式：**INCLUDE 宏库文件名**

该伪指令一般放在程序的最前面。宏库的展开情况，可以通过查看汇编过程中生成的.LST 列表文件。

建立宏库以后，宏库文件可以为编写不同的程序所共享，使得编写汇编语言程序与高级语言的编程类似。

13.1.6 宏指令与子程序

宏指令与子程序都可以用来处理程序中重复使用的程序段，使程序结构简洁，易于阅读和理解。但是，它们是完全不同的两个概念，有着本质的区别，不同点体现在以下几个方面：

(1) 处理时间不同：宏指令在程序汇编过程中由宏汇编程序处理，而子程序是在目标程序执行时，由 CPU 直接执行。

(2) 调用方式不同：宏调用是在汇编时用宏体替换宏调用指令，进行宏展开。宏调用多少次就要展开多少次，因此宏指令不会缩短目标代码长度，占用内存空间大。而子程序有专门的调用指令 CALL 和返回指令 RET，每调用一次子程序，CALL 就执行一次，汇编程序生成的只是 CALL 指令对应的目标代码，子程序的目标代码只出现一次，因此目标代码短，占用内存空间少。

(3) 参数传递方式不同：宏调用采用虚实结合的方法实现参数传递，执行效率高；而子程序是利用寄存器、存储器或堆栈等传递参数，参数传递需要专门的指令完成，要花费额外开销。

(4) 执行速度不同：子程序调用需要执行 CALL 和 RET 指令，需要专门的指令传递参数，需要保护现场和恢复现场，因此执行速度慢。宏指令不存在这些问题，因此执行速度快。

在实际应用中，是采用子程序还是宏指令，需要从多个方面权衡。一般情况下，当程序较短或对执行速度要求较高时，采用宏指令编写程序；当程序段较长或者要求目标代码占

用较少存储空间时，适宜采用子程序编程。

13.2　重复汇编

重复汇编伪指令可以出现在宏定义中，也可以单独出现在程序中。重复汇编是在程序汇编期间对某些语句序列进行重复的汇编，而不是在程序运行期间执行重复操作。

13.2.1　重复汇编伪操作

格式：REPT　表达式
　　　⋮　(重复块)
　　　ENDM

功能：表达式的值用来确定重复块的重复汇编的次数。

【例 13-5】　重复汇编例子。

```
DATA SEGMENT
   M = 0
   NUM = 4
   REPT  5
      M = M + 1
      DB  NUM * M
   ENDM
DATA ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       ;此处输入代码段代码
       MOV AH,4CH
       INT 21H
CODE ENDS
       END START
```

重复汇编伪指令在汇编后，把 04H、08H、0CH、10H、14H 分配给五个连续的字节单元中。数据段的定义情况如图 13-2 所示。

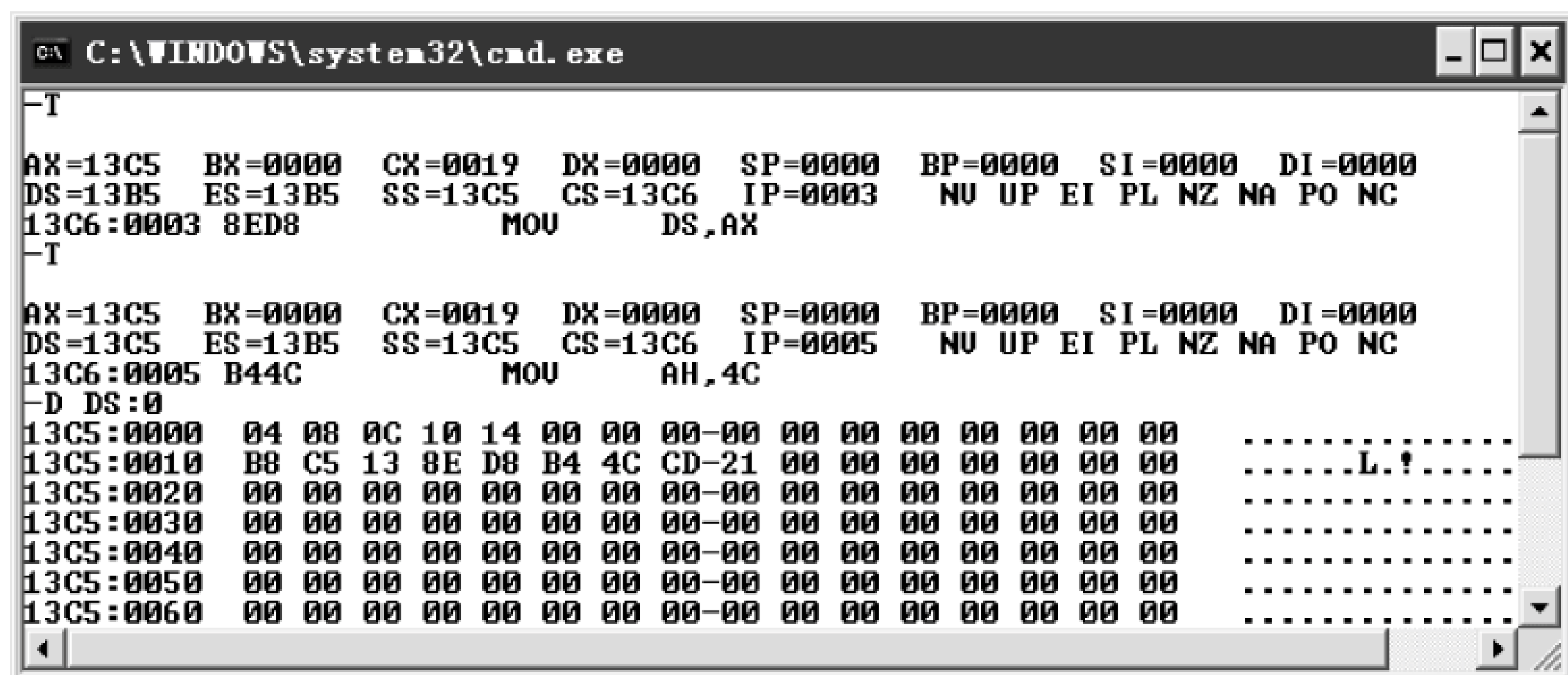


图 13-2　重复汇编伪指令实现的数据段定义

13.2.2　不定次数的重复汇编伪操作

格式：IRP　形参,<实参 1,实参 2,…>

　　⋮(重复块)

　　ENDM

功能：IRP 与 ENDM 之间重复块的重复次数由实参的个数所确定。每次重复汇编语句序列时,用一个实参取代形参。第一次用实参 1,第二次用实参 2,直到实参使用完为止。

【例 13-6】　利用 IRP/ENDM 改写例 13-5 重复汇编的例子。

```
DATA SEGMENT
   NUM = 4
   IRP M,< 1,2,3,4,5 >
      DB   NUM * M
   ENDM
DATA ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       ;此处输入代码段代码
       MOV AH,4CH
       INT 21H
CODE ENDS
       END START
```

13.2.3　IRPC 不定次数的重复字符伪操作

格式：IRPC　形参,字符串(或<字符串>)

　　⋮(重复块)

　　ENDM

功能：重复的次数由字符串的长度确定。宏汇编程序将把重复块连续地重复汇编,每次重复时依次用字符串中的一个字符作为实参代替重复块中的形参。如果字符串含有空格、逗号等分隔符时,那么字符串就需要用一对尖括号括起来；否则,可以不用尖括号。

【例 13-7】　利用 IRPC/ENDM 改写例 13-5 重复汇编的例子。

```
DATA SEGMENT
   NUM = 4
   IRPC M,12345
      DB   NUM * M
   ENDM
DATA ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       ;此处输入代码段代码
       MOV AH,4CH
       INT 21H
```

```
CODE ENDS
        END START
```

13.3 条件汇编

使用条件汇编伪指令，可以使宏汇编语言程序只对某些部分程序进行汇编。条件汇编仅在宏汇编程序的汇编期间对条件进行判断，是否进行汇编，而不是在程序执行期间进行判断。

条件汇编语句的基本格式：

```
IF  表达式
⋮}程序段 1
[ELSE
⋮}程序段 2]
ENDIF
```

功能：当表达式的值不为 0 时，条件为真，对程序段 1 进行汇编。当条件为假时，对程序段 1 就不再进行汇编，如果存在程序段 2，则对程序段 2 进行汇编。

条件汇编伪指令可概括在表 13-1 中，使用方式与 IF 类似，在此不予赘述。

表 13-1　条件汇编伪指令

格　　式	功　　能
IF 表达式	当表达式的值不为 0 时，条件为真
IFE 表达式	当表达式的值为 0 时，条件为真
IFDEF 符号	当符号已定义时，条件为真
IFNDEF 符号	当符号未定义时，条件为真
IFB <参数>	当参数为空格时，条件为真
IFNB <参数>	当参数不为空格时，条件为真
IFIDN <参数 1>,<参数 2>	当参数 1 和参数 2 相同时，条件为真
IFDIF <参数 1>,<参数 2>	当参数 1 和参数 2 不相同时，条件为真
IF1	当汇编处于第一次扫描时，条件为真
IF2	当汇编处于第二次扫描时，条件为真

【例 13-8】 输出一个字符的宏定义：无参数时输出'＊'，有参数时输出参数指定的字符。

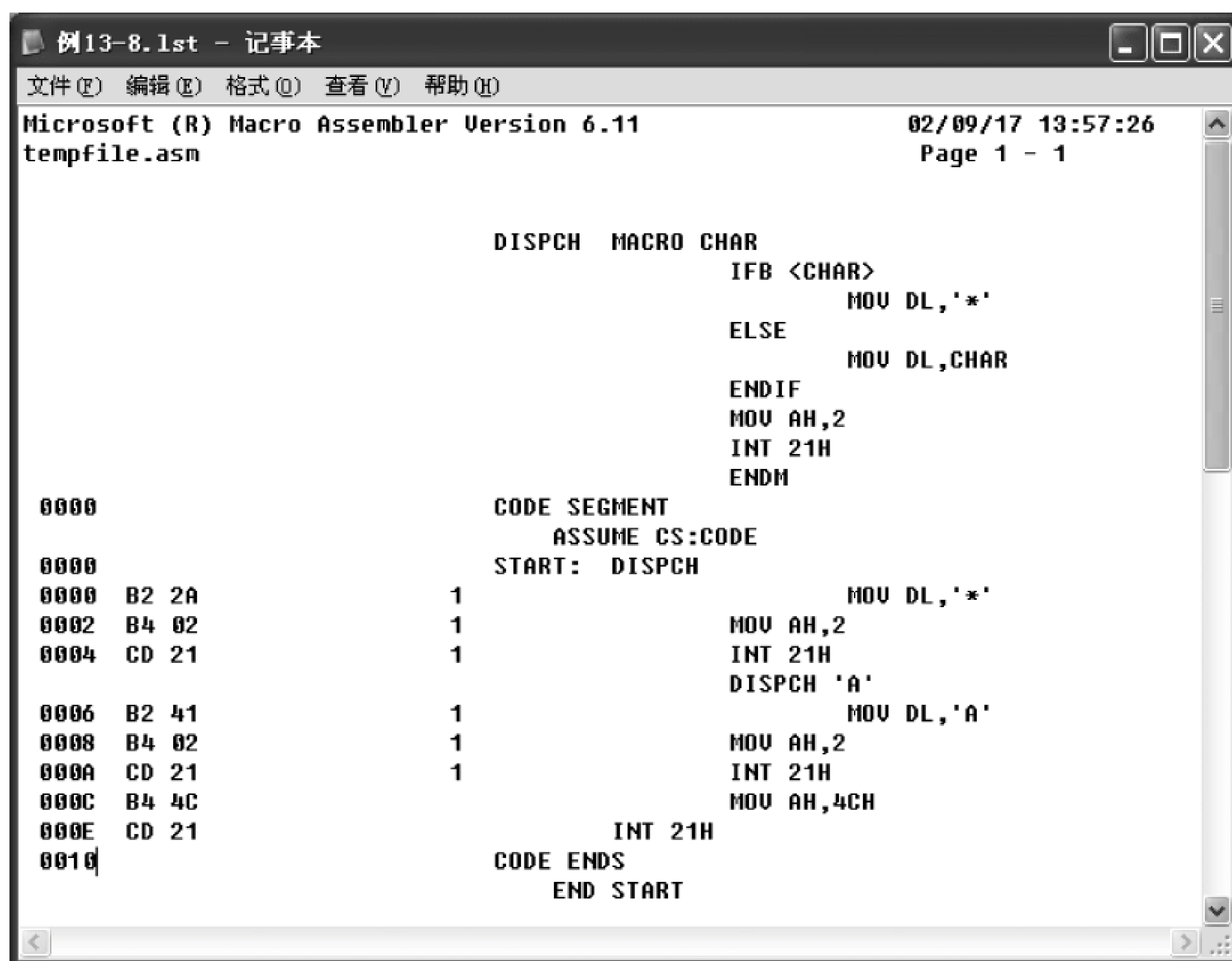
主程序中有两个宏调用，分别是带参数和不带参数，注意观察汇编时宏调用被展开的情况，体会条件汇编伪指令的作用。

程序如下：

```
DISPCH  MACRO CHAR
            IFB <CHAR>
                MOV DL,'*'
            ELSE
                MOV DL,CHAR
            ENDIF
```

```
            MOV AH,2
            INT 21H
            ENDM
CODE    SEGMENT
    ASSUME CS:CODE
START:      DISPCH                  ;输出'*'
            DISPCH 'A'              ;输出'A'
            MOV AH,4CH
            INT 21H
CODE    ENDS
    END START
```

打开汇编时生成的.LST 文件,可以看出宏展开的情况,如图 13-3 所示。DISPCH 处已经被“MOV DL,'*'”替代,DISPCH 'A'处被“MOV DL, 'A'”替代。



```
例13-8.lst - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Microsoft (R) Macro Assembler Version 6.11              02/09/17 13:57:26
tempfile.asm                                              Page 1 - 1


                                DISPCH  MACRO CHAR
                                        IFB <CHAR>
                                                MOV DL,'*'
                                        ELSE
                                                MOV DL,CHAR
                                        ENDIF
                                        MOV AH,2
                                        INT 21H
                                        ENDM
 0000                           CODE SEGMENT
                                    ASSUME CS:CODE
 0000                           START:  DISPCH
 0000  B2 2A              1                     MOV DL,'*'
 0002  B4 02              1             MOV AH,2
 0004  CD 21              1             INT 21H
                                        DISPCH 'A'
 0006  B2 41              1                     MOV DL,'A'
 0008  B4 02              1             MOV AH,2
 000A  CD 21              1             INT 21H
 000C  B4 4C                            MOV AH,4CH
 000E  CD 21                    INT 21H
 0010                           CODE ENDS
                                    END START
```

图 13-3 输出字符的宏展开情况

13.4 实验内容

【实验目的】 通过上机操作理解宏定义、宏调用、宏展开的概念,熟练运用宏功能编写应用程序。

【实验 13-1】 定义宏指令 SEND:要求把存储器中的一个用'$'字符结尾的字符串传送到另一个存储区中。然后调用宏指令 SEND,实现将 BLOCK1 中的一个字符串传送到 BLOCK2 中,注意观察汇编时宏调用被展开的情况。

程序如下:

```
;宏定义 SEND
```

```
SEND MACRO SRCS, DSTS
        LOCAL NEXT,EXIT
        PUSH AX
        PUSH SI
        MOV SI, 0
NEXT:   MOV AL, SRCS[SI]
        MOV DSTS[SI], AL
        CMP AL, '$ '
        JZ EXIT
        INC SI
        JMP NEXT
EXIT:   POP SI
        POP AX
        ENDM
;主程序
DATA SEGMENT
    BLOCK1 DB 'ABCDEFGHIJKLMNOP$ '
    BLOCK2 DB 20 DUP(?)
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
        MOV DS,AX
        SEND BLOCK1,BLOCK2
        MOV AH,4CH
        INT 21H
CODE ENDS
        END START
```

【实验 13-2】 从键盘输入一个字符串(只含有英文字母),先将它原样显示一遍,然后将其中的小写字母转换为大写字母后再显示一遍。要求:显示字符串的功能和小写字母转换成大写字母均采用宏功能实现,注意观察汇编时宏调用被展开的情况。

(1) 先定义三个宏,分别是显示字符串 DISPSTR、小写字母转换成大写字母 LOWUP、回车换行 NEWLINE,然后再编写主程序,利用宏调用实现。程序如下:

```
;字符串输出,COUNT 字符串长度,STRING 字符串起始地址
DISPSTR  MACRO COUNT,STRING
        LOCAL NEXT
        PUSH BX
        PUSH CX
        MOV CL,COUNT                        ;CL←字符串长度
        MOV CH,0
        LEA BX,STRING                       ;字符串存储的起始地址
NEXT:   MOV AH,02H
        MOV DL,[BX]
        INT 21H                             ;输出一个字符
        INC BX
        LOOP NEXT
        POP CX
        POP BX
```

```
        ENDM
;小写字母转换成大写字母
LOWUP  MACRO COUNT,STRING
        LOCAL LTOU,NEXT
        MOV SI,OFFSET STRING
        MOV CL,COUNT
        MOV CH,0
LTOU:   MOV AL,[SI]
        CMP AL,'a'
        JB NEXT
        CMP AL,'z'
        JA NEXT
        SUB [SI],'a'-'A'
NEXT:   INC SI
        LOOP LTOU
        ENDM
;回车换行
NEWLINE   MACRO
        MOV AH,02H
        MOV DL,0DH
        INT 21H
        MOV DL,0AH
        INT 21H
        ENDM
;主程序
DATA SEGMENT
    BUFFER DB 20,?,20 DUP(?)
DATA ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:DATA
START:MOV AX,DATA
        MOV DS,AX
        MOV AH,0AH
        MOV DX,OFFSET BUFFER
        INT 21H                          ;接收用户输入字符串
        NEWLINE
        DISPSTR BUFFER+1,BUFFER+2        ;原样输出
        NEWLINE
        LOWUP BUFFER+1,BUFFER+2          ;小写转换成大写
        DISPSTR BUFFER+1,BUFFER+2        ;输出
        MOV AH,4CH
        INT 21H
CODE ENDS
    END START
```

（2）定义一个宏库文件“实验 13-2.MAC”，把三个宏定义放在一个宏库文件中，然后调用宏库文件编写主程序实现。程序如下：

```
INCLUDE 实验 13-2.MAC
DATA SEGMENT
    BUFFER DB 20,?,20 DUP(?)
```

```
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:MOV AX,DATA
      MOV DS,AX
      MOV AH,0AH
      MOV DX,OFFSET BUFFER
      INT 21H                          ;接收用户输入字符串
      NEWLINE
      DISPSTR BUFFER+1,BUFFER+2        ;原样输出
      NEWLINE
      LOWUP BUFFER+1,BUFFER+2          ;小写转换成大写后输出
      DISPSTR BUFFER+1,BUFFER+2
      MOV AH,4CH
      INT 21H
CODE ENDS
   END START
```

习　　题

1. 分别解释宏定义、宏调用、宏展开的概念。

2. 宏与子程序有什么区别?

3. 分别定义如下宏指令,然后调用宏指令实现将键盘输入的大写字母转换为相应的小写字母后显示输出。

(1) 宏指令 LCASE,实现将大写字母变为对应的小写字母。

(2) 宏指令 GETCHAR,实现从键盘接收一个字符。

(3) 宏指令 PUTCHAR,实现在显示器上输出一个字符。

(4) 宏指令 EXIT,结束程序返回 DOS 提示符下。

第 14 章

综合性程序设计案例

本章是以几个典型的综合性程序设计案例,引导学生开展综合性汇编语言程序设计,要求在学习前面各章知识的基础上,能够融会贯通,全面掌握汇编语言程序设计的方法,强化综合性编程与调试的能力。

14.1 十进制数的加法程序

【例 14-1】 从键盘输入两个一位十进制数,做加法运算,并显示十进制运算的结果。要求:键盘输入和输出结果之前,都先显示提示信息。

(1) 所有功能都采用主程序形式实现。

(2) 定义宏指令 DISPSTR:实现显示字符串的功能;定义宏指令 BCDASC:实现非压缩 BCD 码转换成 ASCII 码后显示输出;定义宏指令 PUTCHAR:输出一个指定字符。调用已定义的宏指令实现,并注意观察汇编时宏展开的情况。

分析:从键盘输入的数字,通过 1 号 DOS 功能调用得到的是数字字符的 ASCII 码,要转换成非压缩 BCD 码才能参加运算,运算结果要使用 DAA 指令。显示提示信息可采用 9 号 DOS 功能调用实现。

(1) 所有功能都采用主程序形式实现。程序如下:

```
DATA SEGMENT
    STR1 DB 0DH,0AH,'PLS INPUT DATA1:$'
    STR2 DB 0DH,0AH,'PLS INPUT DATA2:$'
    STR3 DB 0DH,0AH,'THE RESULT:$'
    D1 DB ?
    D2 DB ?
DATA ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA
START:    MOV   AX,DATA
          MOV   DS,AX
INPUT1:   MOV   DX,OFFSET STR1
          MOV   AH,9
          INT   21H                      ;显示提示信息 STR1
          MOV   AH,1
          INT   21H                      ;输入一个数字(0～9)
          AND   AL,0FH                   ;ASCII 码转换为非压缩 BCD
          MOV   D1,AL                    ;第一个数存入 D1 单元
INPUT2:   MOV   DX,OFFSET STR2
```

```
            MOV   AH,9
            INT   21H                       ;显示提示信息 STR2
            MOV   AH,1                      ;输入第二个数字(0～9)
            INT   21H
            AND   AL,0FH                    ;ASCII 码转换为非压缩 BCD
            MOV   D2,AL                     ;第二个数存入 D2 单元
DISP1:      MOV   DX,OFFSET STR3
            MOV   AH,9
            INT   21H                       ;显示提示信息 STR3
            MOV   AL,D1
            ADD   AL,D2                     ;计算 D1 + D2
            DAA                             ;十进制数调整
            MOV   BL, AL
            CMP   AL,10H
            JB    DISP2                     ;若小于 10,就输出个位数
            MOV   DL, BL                    ;否则,将高四位移到低四位
            MOV   CL, 4
            SHR   DL,   CL
            ADD   DL, 30H                   ;十位数的 BCD 变成 ASCII 码
            MOV   AH, 2
            INT   21H                       ;显示十位数上的数字
DISP2:      MOV   DL, BL
            AND   DL,0FH
            ADD   DL,30H
            MOV   AH, 2
            INT   21H                       ;显示个位数上的数字
            MOV   AH,4CH
            INT   21H
CODE    ENDS
        END START
```

在集成实验环境下运行,输入 5 和 7,则计算结果显示为 12,如图 14-1 所示。



图 14-1　两个一位十进制数加法的运行结果

(2) 由于宏定义较多,不妨把三个宏定义放在一个宏库文件中,然后调用宏库文件编写主程序实现。程序如下:

```
;宏库文件名: 例 14 - 1.MAC
;显示输出字符串,以'$'为结束标志
DISPSTR   MACRO   STRING
        LEA DX,STRING
        MOV AH,9
        INT 21H
        ENDM
;非压缩 BCD 码转换成 ASCII 码后显示输出
```

```
BCDASC  MACRO BCD
        PUSH AX
        PUSH BX
MOV     DL,BCD
        ADD DL,30H
        MOV AH,2
        INT 21H
        POP BX
        POP AX
        ENDM
;显示输出一个字符
PUTCHAR  MACRO CHAR
        MOV DL,CHAR
        MOV AH,2
        INT 21H
        ENDM

;主程序
INCLUDE 例 14-1.MAC                ;调用宏库文件
DATA SEGMENT
    STR1 DB 0DH,0AH,'PLS INPUT DATA1: $ '
    STR2 DB 0DH,0AH,'PLS INPUT DATA2: $ '
    STR3 DB 0DH,0AH,'THE RESULT: $ '
    D1    DB ?
    D2    DB ?
DATA ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
INPUT1: DISPSTR STR1               ;显示提示信息 STR1
        MOV AH,1
        INT 21H                    ;输入一个数字(0~9)
        AND AL,0FH                 ;ASCII 码转换为非压缩 BCD
        MOV D1,AL                  ;第一个数存入 D1 单元
INPUT2: DISPSTR STR2               ;显示提示信息 STR2
        MOV AH,1                   ;输入第二个数字(0~9)
        INT 21H
        AND AL,0FH                 ;ASCII 码转换为非压缩 BCD
        MOV D2,AL                  ;第二个数存入 D2 单元
DISP1:  DISPSTR STR3               ;显示提示信息 STR3
        BCDASC D1
        PUTCHAR '+'
        BCDASC D2
        PUTCHAR '='
        MOV AL,D1
        ADD AL,D2                  ;计算 D1+D2
        DAA                        ;十进制数调整
        MOV BL, AL
        CMP AL,10H
        JB DISP2                   ;若小于 10,就输出个位数
```

```
         MOV DL, BL                  ;否则,将高四位移到低四位
         MOV CL, 4
         SHR DL, CL
         BCDASC DL                   ;显示十位数上的数字
DISP2:   MOV DL, BL
         AND DL,0FH
         BCDASC DL                   ;显示个位数上的数字
         MOV AH,4CH
         INT 21H
CODE ENDS
    END START
```

在集成实验环境下运行,输入 5 和 7,则计算结果显示如图 14-2 所示。

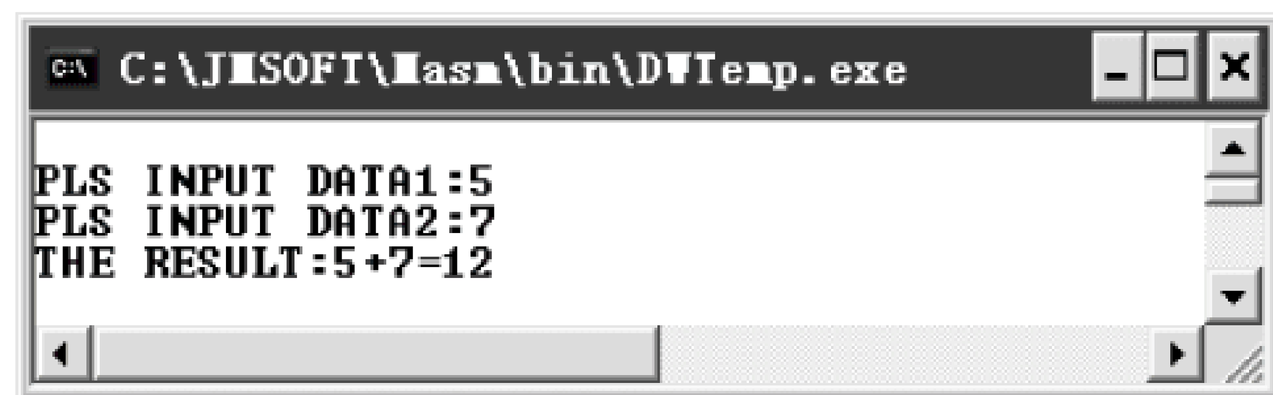


图 14-2　改进后加法程序的运行结果

思考:本程序适合于做两个一位十进制数的加法,如果完成以下问题应如何实现?

(1) 两位十进制数减去一位十进制数,例如 12-4=08。

(2) 两个一位十进制数的乘法,例如 3*7=21。

(3) 两位十进制数除以一位十进制数,例如 55/5=11。

14.2　九九乘法表输出程序

【例 14-2】 在屏幕上按照如下格式输出乘法表:

```
1*1=1
1*2=2   2*2=4
1*3=3   2*3=6   3*3=9
          ⋮
1*9=9   2*9=18  3*9=27      …       9*9=81
```

分析:九九乘法表输出程序是一个双重循环结构的程序,外层循环用 BL 寄存器保存并记录,外循环每执行一次,BL 的值加 1。内循环用 BH 寄存器保存并记录,如果 BH 的值小于 BL 的值则执行内循环,内循环中每和 BL 比较一次,成立则其值加 1,不成立则换行,同时也比较外循环是否继续满足条件执行,即比较 BL 的值是否小于 10,小于就执行外循环,否则就结束程序。

在程序设计中,DIS 子程序把内存中的十六进制数转换成十进制数。还有两个宏指令:BCDASC 完成把十进制数的非压缩 BCD 码转换成 ASCII 码后显示输出;PUTCHAR 实现显示输出一个字符。

程序如下:

```
;非压缩 BCD 码转换成 ASCII 码后显示输出
```

```
BCDASC  MACRO BCD
        PUSH AX
        PUSH BX
        MOV DL,BCD
        ADD DL,30H
        MOV AH,2
        INT 21H
        POP BX
        POP AX
        ENDM
;显示输出一个字符
PUTCHAR  MACRO CHAR
        MOV DL,CHAR
        MOV AH,2
        INT 21H
        ENDM
;主程序
DATA   SEGMENT
  TABLE DW 81 DUP(0)
DATA   ENDS
CODE   SEGMENT
   ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
        MOV DS,AX
        LEA DI,TABLE                    ;DI 指向 TABLE 的首地址
        MOV BL,1                        ;外循环值
MUL1:  MOV BH,1                        ;内循环值
MUL2:  PUSH CX                         ;压栈
        BCDASC BL                       ;输出第一个外循环值
        PUTCHAR '*'                     ;输出乘号
        BCDASC BH                       ;输出第一个内循环值
        PUTCHAR '='                     ;输出等号
        MOV AL,BH                       ;把 BH 的值赋给 AL 寄存器中
        MUL BL                          ;乘法指令,AL*BL,结果放入 AL 中
        MOV [DI],AL                     ;DI 指向乘的结果
        CALL DIS                        ;调用子函数 DIS
        PUTCHAR 20H                     ;输出空格
        PUTCHAR 20H
        ADD DI,2                        ;DI 指向它的下一个地址
        POP CX
        INC BH
        CMP BH,BL                       ;BH 小于 BL 时,转移至 MUL2
        JBE MUL2
        PUTCHAR 0DH
        PUTCHAR 0AH                     ;输出回车换行
        INC BL
        CMP BL,10                       ;BL 小于 10 时,转移至 MUL1
        JB MUL1
        JMP EXIT
;DIS 功能: 乘积的结果大于 10,则需要转化成十进制数
DIS     PROC   NEAR
```

```
        PUSH AX
        PUSH DX
        MOV DH,10
        CMP AX,10
        JB NEXT1                   ;AX 的值低于 10 转向 NEXT1
        DIV DH                     ;否则,除以 10,结果存在 AX 中
        BCDASC AL
        MOV AL,AH
NEXT1:  BCDASC AL
        POP DX
        POP AX
        RET
DIS     ENDP
EXIT:   MOV AH,4CH
        INT 21H
CODE   ENDS
        END START
```

在集成实验环境下运行,输出结果如图 14-3 所示。

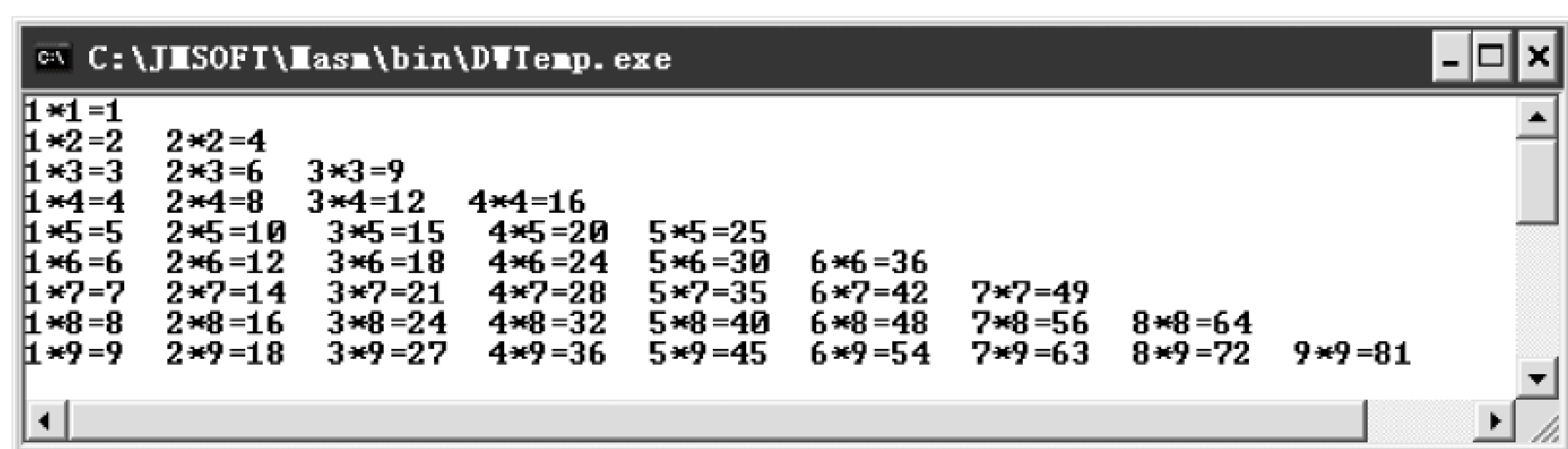


图 14-3　九九乘法表输出结果

思考:九九乘法表按以下几种不同的格式输出,应该如何实现?

```
(1) 1*1=1
    2*1=2   2*2=4
    3*1=3   3*2=6    3*3=9
              ⋮
    9*1=9   9*2=18   9*3=27   …   9*9=81
(2) 1*1=1   1*2=2    1*3=3    …   1*9=9
            2*2=4    2*3=6    …   2*9=18
                     3*3=9    …   3*9=27
                                    ⋮
                                  9*9=81
```

14.3　代码转换程序

【例 14-3】　将从键盘输入的多位十进制数转换为 16 位二进制数并显示在屏幕上,要求十进制数为 0～65 535。

分析:从键盘输入的十进制数,是以字符串的形式输入的,存储时是 ASCII 码。所以,

本题的关键是 ASCII 码转换为二进制数。为此，设计了一个子程序 DECIBIN 完成十进制数转换为二进制数，分两步进行：①十进制数的 ASCII 码转换为非压缩 BCD 码；②BCD 码转换为二进制数。

程序如下：

```
;宏定义 DISPSTR: 显示输出字符串,以'$'为结束标志
DISPSTR  MACRO  STRING
      LEA DX,STRING
      MOV AH,9
      INT 21H
      ENDM
;主程序
DATA  SEGMENT
  INPROMPT   DB 0DH, 0AH,'Input the decimal number (0～65535):$'
  OUTPROMPT  DB 0DH, 0AH,'Output the binary number:$'
  BUFFER     DB 6,?,6 DUP(?)
DATA  ENDS
CODE  SEGMENT
   ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        DISPSTR INPROMPT                ;显示输入提示信息
        MOV AH,0AH
        LEA DX,BUFFER
        INT 21H                         ;输入数码
        MOV CL,BUFFER + 1
        MOV CH,0
        LEA BX,BUFFER + 2
        CALL DECIBIN
        MOV BX,AX                       ;AX 中的二进制数值存到 BX
        DISPSTR OUTPROMPT               ;显示输出提示
        MOV CX,16                       ;16 表示二进制数的位数
BINPRT: ROL BX,1                        ;BX 的最高位循环移位到 D0 位
        MOV DL,BL
        AND DL,01H
        ADD DL,30H                      ;把一位二进制数转换为 ASCII 码
        MOV AH,2
        INT 21H
        LOOP BINPRT
        MOV AH,4CH                      ;返回到 DOS
        INT 21H
;子程序 DECIBIN
;功能：十进制数码转换为二进制数值的过程
;入口参数：CX←数码串长度,BX←数码缓冲区首地址
;出口参数：转换后的二进制数值在 AX 中
DECIBIN  PROC NEAR
        XOR AX,AX                       ;AX 清 0
DTOB:   MOV DX,10
        MUL DX                          ;高位乘 10
        MOV DL,[BX]                     ;读取低位 ASCII 码
```

```
            INC BX                          ;BX指向下一个字符
            AND DL,0FH                      ;字符ASCII转换为BCD
            XOR DH,DH
            ADD AX,DX                       ;加低位
            LOOP DTOB
            RET
DECIBIN  ENDP
CODE  ENDS
   END START
```

在集成实验环境下运行,输入十进制数 65 534,则输出二进制数为 1111111111111110,如图 14-4 所示。

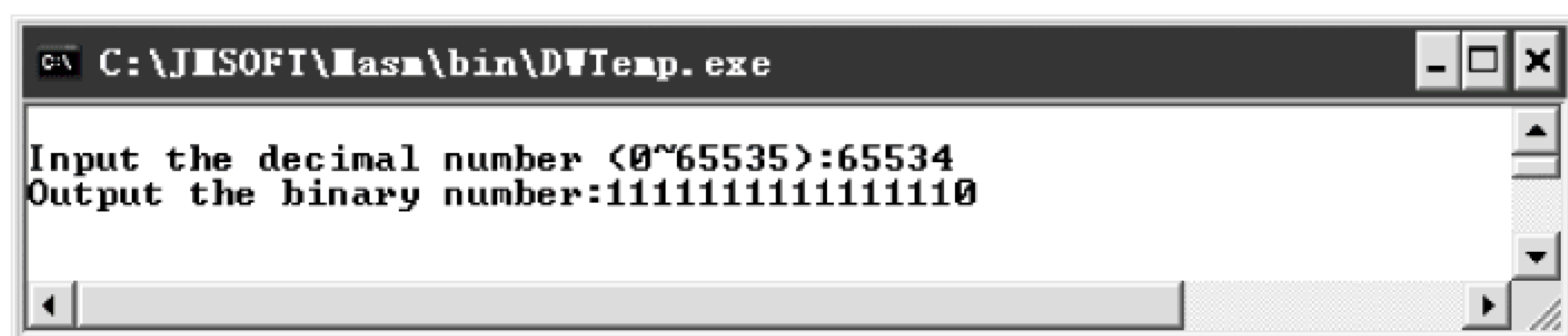


图 14-4　代码转换程序运行结果

思考：如何实现以下代码转换?

(1) 十进制数转换为十六进制数。

(2) 二进制数转换为十进制数。

(3) 十六进制数转换为十进制数。

14.4　菜 单 程 序

【例 14-4】 设计一个菜单程序,实现四个功能:

(1) 大小写转换:输入一个字符串(长度小于 20),将其中的大小写字母互换,其他字符不转换;

(2) 十进制乘法:输入两个一位十进制数,做乘法运算,乘积显示在屏幕上;

(3) 字符串比较:分别输入两个长度相同的字符串,比较两个字符串是否相同,若不同,则需要说明相同的字符个数和不同的字符个数;

(4) 退出:结束程序返回到 DOS 提示符下。

分析:在数据段中定义菜单项,可以采用 9 号 DOS 功能调用实现。然后,根据用户输入的选项执行某段程序。采用一般的分支结构程序设计方法,如果用户输入选项序号"1",就执行标号 PRG1 处的程序段;如果输入选项序号"2",就执行标号 PRG2 处的程序段,依次类推。这种方法对于菜单项较少的情况,简单快捷。

在程序设计中,把经常用到的相对独立的功能,都定义为宏指令,这样便于阅读主程序。由于宏定义较多,因此先建立了一个宏库文件 MENUPRG.MAC,把所有的宏定义放在一个文件里。

```
;*****************************************************
;宏库文件名:MENUPRG.MAC
;*****************************************************
```

```
;显示输出以'$'为结束标志的字符串,9 号 DOS 功能调用
DISPSTR1  MACRO  STRING
          LEA DX,STRING
          MOV AH,9
          INT 21H
          ENDM
;字符串输出,COUNT 字符串长度,STRING 字符串起始地址
DISPSTR2  MACRO COUNT,STRING
          LOCAL NEXT
          PUSH BX
          PUSH CX
          MOV CL,COUNT              ;CL←字符串长度
          MOV CH,0
          LEA BX,STRING             ;字符串存储的起始地址
NEXT:     MOV AH,02H
          MOV DL,[BX]
          INT 21H                   ;输出一个字符
          INC BX
          LOOP NEXT
          POP CX
          POP BX
          ENDM
;大小写字母互换,其他字符不转换
EXLOWUP   MACRO COUNT,STRING
          LOCAL LTOU,NEXT,UPPER
          MOV SI,OFFSET STRING
          MOV CL,COUNT
          MOV CH,0
LTOU:     MOV AL,[SI]
          CMP AL,'a'
          JB UPPER
          CMP AL,'z'
          JA NEXT
          SUB [SI],'a'-'A'          ;小写字母变为大写字母
          JMP NEXT
UPPER:    CMP AL,'A'
          JB NEXT
          CMP AL,'Z'
          JA NEXT
          ADD [SI],'a'-'A'          ;大写字母变为小写字母
NEXT:     INC SI
          LOOP LTOU
          ENDM
;回车换行
NEWLINE   MACRO
          MOV AH,02H
          MOV DL,0DH
          INT 21H
          MOV DL,0AH
          INT 21H
          ENDM
```

```
;显示输出一个字符,2 号 DOS 功能调用
PUTCHAR    MACRO CHAR
           MOV DL,CHAR
           MOV AH,2
           INT 21H
           ENDM
;字符串输入,0AH 号 DOS 功能调用
GETSTR     MACRO BUFFER
           LEA DX,BUFFER
           MOV AH,0AH
           INT 21H
           ENDM

;*****************************************************
;菜单主程序: 例 14-4.ASM
;*****************************************************
INCLUDE MENUPRG.MAC                ;使用宏库
DATA SEGMENT
  MENUITEM  DB 0DH,0AH,'      1.Case Swaps'
            DB 0DH,0AH,'      2.Decimal Multiplication'
            DB 0DH,0AH,'      3.String Comparison'
            DB 0DH,0AH,'      4.Exit'
            DB 0DH,0AH,'   Select Menu Item: $ '
    STR1    DB 0DH,0AH,'Input string1: $ '
    STR2    DB 0DH,0AH,'Input string2: $ '
    STRING1 DB 20,?,20 DUP(?)
    STRING2 DB 20,?,20 DUP(?)
    BUFFER  DB 20,?,20 DUP(?)
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       DISPSTR1 MENUITEM          ;显示菜单,选择菜单项序号
       MOV AH,1                   ;输入选择的菜单项序号
       INT 21H
       CMP AL,'1'
       JZ PRG1
       CMP AL,'2'
       JZ PRG2
       CMP AL,'3'
       JZ PRG3
       CMP AL,'4'
       JZ PRG4
;*********************************************
;功能 1: 输入一个字符串(长度小于 20),将其中的大小写字母互换
;*********************************************
 PRG1: NEWLINE
       GETSTR BUFFER
       NEWLINE
       EXLOWUP BUFFER+1,BUFFER+2        ;大小写互换
```

```
        DISPSTR2 BUFFER + 1,BUFFER + 2        ;输出转换后结果
        JMP PRG4
;*********************************************
;功能 2: 两个一位十进制数的乘法主程序
;*********************************************
PRG2:   MOV CX,2
        MOV BL,1
NEXT2:  MOV AH,1
        INT 21H
        SUB AL,30H
        JC NEXT2
        CMP AL,09H
        JA NEXT2
        MUL BL
        AAM
        MOV BL,AL
        PUSH AX
        PUTCHAR 0DH
        PUTCHAR 0AH
        LOOP NEXT2
        POP AX
DISP:   MOV BX,AX
        MOV DL,BH
        ADD DL,30H
        PUTCHAR DL                            ;输出高位
        MOV DL,BL
        ADD DL,30H
        PUTCHAR DL                            ;输出低位
        JMP PRG4
;*****************************************
;功能 3: 字符串比较,统计相同字符数和不同字符数
;*****************************************
PRG3:   DISPSTR1 STR1
        GETSTR STRING1
        DISPSTR1 STR2
        GETSTR STRING2
        NEWLINE
        LEA SI,STRING1 + 2
        LEA DI,STRING2 + 2
        MOV BX,0                              ;BH 记录相同的个数,BL 记录不同的个数
        MOV CL,STRING1 + 1
        MOV CH,0
CMPSTR:MOV AL,[SI]
        CMP AL,[DI]                           ;两个对应字符比较
        JZ EQUL                               ;相同转 EQUL
        INC BL                                ;不同 BL 加 1
        JMP NEXT3
EQUL:   INC BH                                ;相同 BH 加 1
NEXT3:  INC SI
        INC DI
        LOOP CMPSTR
```

```
        PUTCHAR 'N'                     ;显示'N',代表不同
        ADD BL,30H                      ;加上 ASCII 码
        PUTCHAR BL                      ;显示不同的个数
        PUTCHAR 'E'                     ;显示'E',代表相同
        ADD BH,30H
        PUTCHAR BH                      ;显示相同的个数
;**************************************
;功能 4: 退出
;**************************************
PRG4:   MOV AH,4CH
        INT 21H
CODE ENDS
        END START
```

在集成实验环境下运行,显示主菜单的效果,如图 14-5 所示。



图 14-5　主菜单

在主菜单中,输入选项序号“1”,则系统进入大小写字母转换子功能程序段,在键盘输入一个字符串"ABCDefgh1234"后,转换结果为"abcdEFGH1234",如图 14-6 所示。

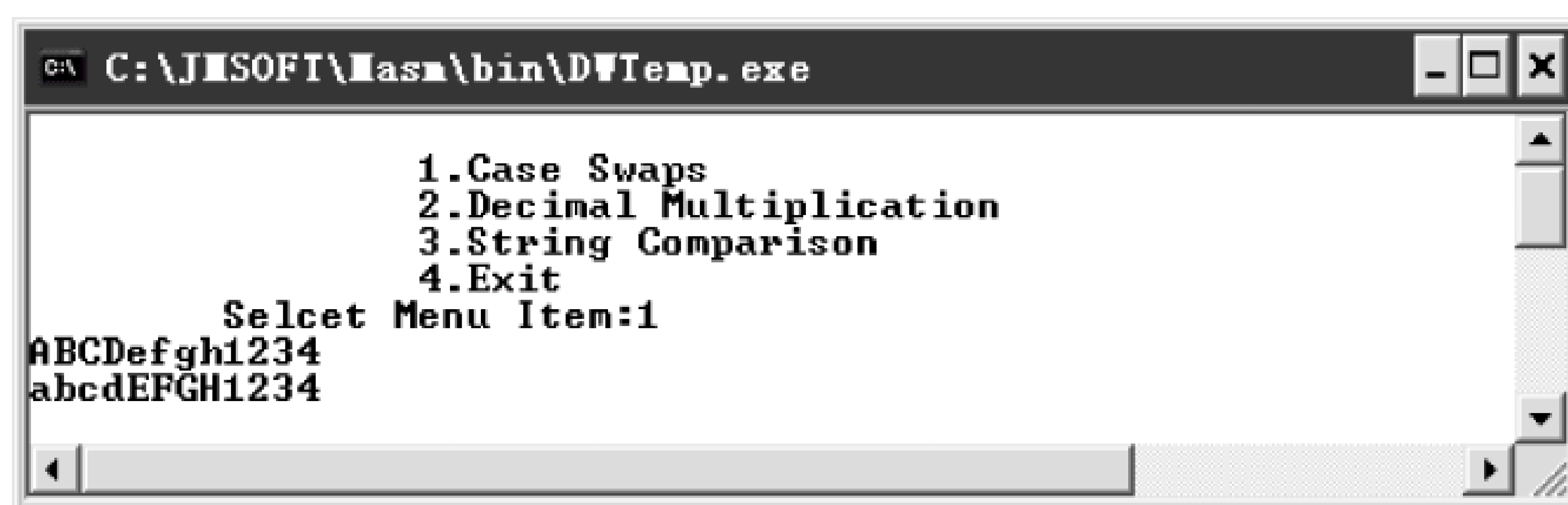


图 14-6　大小写字母转换功能

在主菜单中,输入选项序号“2”,则系统进入两个一位十进制数的乘法子功能程序段,键盘输入一个“3”和一个“8”,结果为“24”,显示效果如图 14-7 所示。

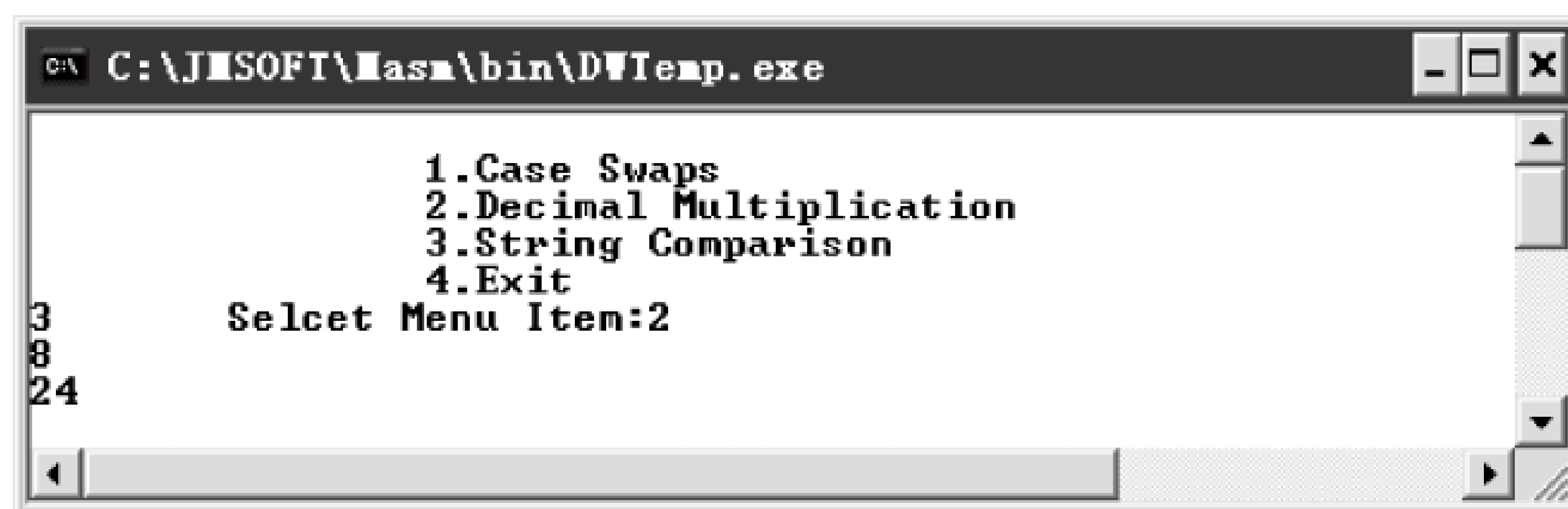


图 14-7　十进制数乘法功能

在主菜单中,输入选项序号“3”,则系统进入字符串比较子功能程序段,在键盘输入两个字符串后,显示“N2E6.”,即 2 个不同字符、6 个相同字符,显示效果如图 14-8 所示。



图 14-8　字符串比较功能

思考：在已有程序的基础上，可以按照自己的意愿和需求，设计并完成菜单程序。如何实现下列功能？

(1) 利用转移表法设计菜单程序，适用于功能较多的情况。

(2) 菜单项的不同功能段设计成子程序，通过调用子程序实现菜单程序。

(3) 采用多模块设计法，每个子功能写成一个程序文件，实现菜单程序，适合于复杂的大型软件开发。

14.5 实验内容

【实验目的】 通过综合性设计实验，学生能全面掌握汇编语言的程序设计方法和上机调试技术，达到融会贯通，能够编写复杂的汇编语言程序，具有良好的应用实践能力。

【实验 14-1】 设计一个简易的整数计算器，能够完成十进制数的加法、减法、乘法、除法的四种运算。菜单形式和主要功能自行确定。

分析：计算器设计的核心是十进制数的加法、减法、乘法、除法的四种运算。从键盘输入的十进制数，需要将该数的 ASCII 码转换为二进制数，才能进行运算，运算结果为二进制数。在显示输出时，需要把二进制数转换成相应的十进制数的 ASCII 码，然后才能显示输出。

为简单起见，数据的取值范围是两个字节的无符号数所能表达的十进制数，无论是参加运算的数，还是运算结果。

参考程序如下：

```
;************************************
;宏定义
;************************************
;显示输出以'$'为结束标志的字符串,9号DOS功能调用
PUTSTR  MACRO  STRING
      LEA DX,STRING
      MOV AH,9
      INT 21H
      ENDM
;回车换行
NEWLINE  MACRO
      MOV AH,02H
      MOV DL,0DH
      INT 21H
```

```
        MOV DL,0AH
        INT 21H
        ENDM
;************************************
;主程序
;************************************
DATA SEGMENT
    INFO0     DB 0DH,0AH,'****************************** $'
    INFO1     DB 0DH,0AH,'Simple integer calculator $'
    INFO2     DB 0DH,0AH,'Software studio $'
    INFO3     DB 0DH,0AH,'$'
    BINFO0    DB 0DH,0AH,'*  1.ADD  * $'
    BINFO1    DB 0DH,0AH,'*  2.SUB  * $'
    BINFO2    DB 0DH,0AH,'*  3.MUL  * $'
    BINFO3    DB 0DH,0AH,'*  4.DIV  * $'
    BINFO4    DB 0DH,0AH,'*  5.EXIT (OR OTHER KEY)  * $'
    CINFO0    DB 0DH,0AH,'*  1.Input the first number  * $'
    CINFO1    DB 0DH,0AH,'*  2.Input the second number  * $'
    CINFO2    DB 0DH,0AH,'*  3.EXIT(OR OTHER KEY)  * $'
    BUFFER    DB 10,?,10 DUP(?)
    ERRMSG    DB 0DH,0AH,'Invalid data! Please try again. $'
    ARRAY1    DB 0DH,0AH,'The first number: $'
    ARRAY2    DB 0DH,0AH,'The second number: $'
    ARRAY     DW 2 DUP(0)
    RESULT    DB 10 DUP(0)
    FLAGS     DB 0
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
MAIN    PROC FAR
START:  PUSH DS
        SUB AX,AX
        PUSH AX
        MOV AX,DATA
        MOV DS,AX
DSPMN:  CALL MAINMENU
        MOV [FLAGS],0FFH
        MOV AH,7                    ;7 号 DOS 功能调用,没有回显
        INT 21H                     ;若采用 1 号 DOS 功能调用,有回显
        CMP AL,31H
        JB QUIT
        CMP AL,34H
        JA QUIT
        MOV [FLAGS],AL              ;记录下输入的菜单项序号
        MOV [ARRAY],0FFH
        MOV [ARRAY+2],0FFH
ADS:    CALL DSPMENU2
        MOV AH,7
        INT 21H
        CMP AL,'1'
        JZ NUM1
```

```
        CMP AL,'2'
        JZ NUM2
        JMP QUIT
NUM1:   MOV DX,OFFSET ARRAY1
        CALL INPUTNUMBER
        CMP CX,0
        JZ QUIT
        MOV [ARRAY],AX
        CALL JISUAN
        JC DSPMN
        JMP ADS
NUM2:   MOV DX,OFFSET ARRAY2
        CALL INPUTNUMBER
        CMP CX,0
        JZ QUIT
        MOV [ARRAY+2],AX
        CALL JISUAN
        JC DSPMN
        JMP ADS
;***********************************
;子程序:ASCBIN
;功能:数字的 ASCII 码转换为二进制数
;***********************************
ASCBIN PROC NEAR
        MOV CH,0
        MOV BX,OFFSET BUFFER
        MOV CL,[BX+1]
        CMP CX,0
        JZ END_P
        PUSH CX
        ADD BX,2
        PUSH BX
L0:     SUB [BX],30H
        INC BX
        LOOP L0
        POP BX
        MOV AX,0
        MOV CH,0
        MOV CL,[BX-1]
L1:     MOV DL,10
        MUL DL
        ADD AL,[BX]
        INC BX
        LOOP L1
        POP CX
END_P: RET
ASCBIN ENDP
;***********************************
;子程序:OUTPUT
;功能:输出结果
;***********************************
```

```
OUTPUT PROC NEAR
        MOV SI,OFFSET RESULT
        ADD SI,4
        MOV [SI+1],'$'
        MOV CX,5
AG1:    MOV DX,0
        MOV BX,10
        DIV BX
        PUSH DX
        ADD DL,30H
        MOV [SI],DL
        DEC SI
        POP DX
        LOOP AG1
        MOV CX,4
AG3:    INC SI
        MOV AL,[SI]
        CMP AL,30H
        JNZ NOZERO
        LOOP AG3
        INC SI
NOZERO:NEWLINE
        MOV DX,SI
        MOV AH,9
        INT 21H
        RET
OUTPUT ENDP
;************************************
;子程序：MAINMENU
;功能：显示主菜单
;************************************
MAINMENU PROC NEAR
        PUTSTR INFO0
        NEWLINE
        PUTSTR INFO1
        PUTSTR INFO2
        PUTSTR BINFO0
        PUTSTR BINFO1
        PUTSTR BINFO2
        PUTSTR BINFO3
        PUTSTR BINFO4
        NEWLINE
        PUTSTR INFO0
        RET
MAINMENU ENDP
;************************************
;子程序：DSPMENU2
;功能：显示二级子菜单
;************************************
DSPMENU2 PROC NEAR
        NEWLINE
```

```
        PUTSTR INFO0
        NEWLINE
        PUTSTR INFO1
        PUTSTR INFO2
        PUTSTR CINFO0
        PUTSTR CINFO1
        PUTSTR CINFO2
        NEWLINE
        PUTSTR INFO0
        RET
DSPMENU2 ENDP
;***********************************
;子程序：INPUTNUMBER
;功能：输入数据
;***********************************
INPUTNUMBER PROC NEAR
        MOV AH,9
        INT 21H
        MOV DX,OFFSET BUFFER
        MOV AH,0AH
        INT 21H
        CALL ASCBIN
        RET
INPUTNUMBER ENDP
;***********************************
;子程序：JISUAN
;功能：四则运算
;***********************************
JISUAN PROC NEAR
        CMP [ARRAY],0FFH
        JZ FANHUI
        CMP [ARRAY+2],0FFH
        JZ FANHUI
        CMP [FLAGS],31H
        JZ ISADD
        CMP [FLAGS],32H
        JZ ISSUB
        CMP [FLAGS],33H
        JZ ISMUL
        JMP ISDIV
FANHUI: CLC
        RET
ISADD:  MOV BX,ARRAY
        MOV AX,ARRAY+2
        ADD AX,BX
        JMP DISP
ISSUB:  MOV AX,ARRAY
        MOV BX,ARRAY+2
        CMP AX,BX
        JB SMALL
        SUB AX,BX
```

```
        JMP DISP
SMALL:  PUTSTR ERRMSG               ;被减数小于减数,错误提示
        JMP ISWRONG
ISMUL:  MOV AX,ARRAY
        MOV BX,ARRAY + 2
        MUL BX
        MOV DX,0
        JMP DISP
ISDIV:  MOV AX,ARRAY
        MOV BX,ARRAY + 2
        MOV DX,0
        DIV BX
        MOV DX,0
DISP:   CALL OUTPUT
ISWRONG:STC
        RET
JISUAN   ENDP
QUIT:   RET
MAIN     ENDP
CODE ENDS
        END START
```

思考：实现更复杂的计算器，可以考虑在数制和功能上拓展。作为课程设计的内容，在本实验的基础上实现以下功能拓展：

(1) 增加计算平方数和立方数的功能。

(2) 计算结果既可以按十进制输出，也可以按十六进制输出。

习　　题

1. 编写子程序：

(1) SUBPRG，输出 BL 个 DL 中的字符；

(2) INPUT，将键盘输入的一位十进制数转换为二进制数。

然后，编写主程序调用上述两个子程序，实现输入 3～9 的一个数 N，输出一个'*'号组成的口字形。要求检查输入数据的有效性，对不合法的数据给出错误输入提示信息。例如，输入 5，则输出：

```
*****
*   *
*   *
*   *
*****
```

2. 补充完善实验 14-1 计算器的功能，拓展内容自定。

附录 A

DEBUG 常用命令

DEBUG 是汇编语言最基础的调试工具,它不仅可以修正汇编语言程序中的错误,而且可以用来编写较短的汇编语言程序。

为便于学习和查阅,现将常用命令汇总如下。

1. 汇编命令 A

格式:A [地址]　　　　　　　　;从指定地址开始输入汇编指令

A 命令用于输入一条或多条汇编语言指令,并把它们汇编成机器代码,存放在从指定地址开始的存储区中。最后按回车键表示指令输入结束。A 命令中如果没有指定地址,则接着上一个 A 命令的最后一个单元开始;若还没有使用过 A 命令,则从当前 CS:IP 所指存储区开始。

如果用 A 命令时未指定地址,则从当前 CS:IP 所指的单元开始存放指令。因为执行程序段时采用默认程序段的段地址,所以在用 A 命令时,一般都不指定地址,这样可以免去设置段地址的操作。

2. 比较命令 C

格式:C <源范围> <目标地址>　　　　;比较内存的两个部分

其中,<源范围>是要比较的内存第一个区域的<起始地址><终止地址>指出的一片连续存储单元,或者由<起始地址> L <长度>指定。

如果两块内存区域内容相同,将不显示任何内容而直接返回到 DEBUG 提示符。如果有差异,DEBUG 将按如下格式显示:

<源地址><源内容><目标内容><目标地址>

3. 显示存储单元命令 D

格式:D [地址]　　　　　　　　;显示当前或指定开始地址的主存内容
　　　D [范围]　　　　　　　　;显示指定范围的主存内容

使用时,[地址]和[范围]是可选项,所以会出现多种形式。如果不指定参数,程序将从以前 D 命令中所指定的地址范围的末尾开始显示 128B 的内容。

4. 修改存储单元命令 E

E 命令用于修改主存单元内容,有以下两种格式。

格式 1:E<地址> <数据表>　　　　;用数据表的数据修改指定单元的内容

格式 2:E<地址　>　　　　　　　;查看指定单元内容后再修改

格式 1 可以使用数据表一次修改一个或多个单元的内容。<数据表>中的内容是要写入每个单元的数据,可以是十六进制数据,也可以是单引号括起来的字符或字符串。数据和

数据间要用空格间隔,数据和字符可以不分隔。

注意:E 命令完成修改后,屏幕上并不会显示执行结果,需要用 D 命令查看修改后的结果。

5. 填充命令 F

F 命令用于对一个主存区域填写内容,这样会改写原来的内容。

格式:F <地址> <范围> <数据表>

该命令将<数据表>中的数据写入从指定起始地址开始的一定范围的主存区域中。如果数据个数超过指定的单元个数范围,则忽略多出的数据项;如果数据个数小于指定的单元个数范围,则重复使用这些数据,直到填满指定的范围区域。

6. 运行命令 G

程序段已经存放到内存后,如果要运行程序段,查看程序段的功能及执行结果,可以使用 G 命令。

格式:G[= 地址] [断点地址 1 [, 断点地址 2 [, … [, 断点地址 10]]]]

G 命令中的地址都是不能指定段地址的,默认为 CS 所指的段。因为只能有一个当前程序段运行,所以执行前要先将 CS 的值设置为要执行的程序段的段地址。等号后的[地址]指定程序段中第一条指令的起始偏移地址。如不指定偏移地址,则从当前 CS:IP 所指的指令开始运行。断点地址指示 G 命令执行时停下来的指令地址,断点可以没有,但最多只能有 10 个。程序会停在第一个断点,后面要继续,仍要使用 G 命令设置新断点。设置多个断点主要是为了在分支结构中能够在分支点停止程序执行。G 命令输入后,从指定地址处开始运行程序,直到遇到设置的断点指令或者程序正常结束,停止执行并显示当前所有寄存器和标志位的内容,以及下一条将要执行的指令(显示内容同 R 命令),以便观察程序运行到此的情况。程序遇到结束指令正常结束,如果是 EXE 文件或 COM 文件在 DEBUG 中执行,将显示“Program terminated normally”。倘若存储区内没有结束指令,则可能会出现死机。所以,使用时一定要确认执行的指令区域,以及是否有结束指令,否则就要指定断点地址。

7. 十六进制运算命令 H

格式:H <参数 1 > <参数 2 >　　　　;对指定的两个参数执行十六进制运算

其中,<参数 1 >和<参数 2 >代表从 0~FFFFH 范围内的任何十六进制数。

H 命令首先将指定的两个参数相加,然后从第一个参数中减去第二个参数。这些计算的结果显示在一行中:先计算和,然后计算差。

8. 端口输入命令 I

格式:I <端口>　　　　;从指定的端口读取并显示一个字节值

9. 读盘命令 L

格式:L <地址> <盘号> <起始逻辑扇区> <所读扇区个数 n>

其中,<地址>的默认值是 CS:100,<盘号>用 0 表示 A 盘,1 表示 B 盘,2 表示硬盘。

L 命令将某个文件或特定磁盘扇区的内容加载到内存。当使用不带参数的 L 命令时,在 DEBUG 命令行上指定的文件将加载到内存中,从地址 CS:100 开始。同时,将 BX 和 CX 寄存器设置为加载的字节数。如果不在 DEBUG 命令行指定文件,所装入的文件将是最近使用 N 命令指定的文件。

例如：将当前盘上的 EXFILE 文件装入 CS:0100 起始的存储区域。

```
-N EXFILE
-L
```

10. 移动命令 M

格式：M <源范围> <目标地址>

其中，<源范围>指定要复制内容的内存区域的起始和终止地址，或起始地址和长度。该命令将一个内存块中的内容复制到另一个内存块中。

11. 名称命令 N

格式：N [驱动器][路径] 文件名

N 命令是为 L 命令指定待加载的文件，或为 W 命令指定写盘文件。

12. 输出命令 O

格式：O <端口> <字节值> ;将字节值发送到输出端口

13. 执行命令 P

P 命令类似于 T 命令，只是执行时不会进入子程序或中断服务程序中。当不需要调试子程序或中断服务程序时，应使用 P 命令而不是 T 命令。

格式：P [=地址] [数值]

P 命令执行循环、重复的字符串指令、软件中断或子例程；或通过任何其他指令跟踪。一般情况下，在调用系统子程序时都不需要跟踪执行结果，这时采用 P 命令最好。对用户自编的子程序，要想调试跟踪子程序的运行状况，还是需要用 T 命令跟踪到子程序内部。

14. 退出命令 Q

格式：Q

Q 命令使 DEBUG 程序退出，返回操作系统的命令提示符状态。

15. 寄存器命令 R

R 命令用于显示和修改处理器中的寄存器，它有三种格式。

格式：

```
R                     ;显示 CPU 内所有寄存器内容和标志寄存器中标志位值
R 寄存器名             ;显示和修改指定的寄存器
RF                    ;显示和修改标志寄存器中的标志位
```

16. 搜索命令 S

格式：S <范围> <字节值列表>

S 命令是在某个地址范围搜索一个或多个字节值的模式。其中，<范围>指定要搜索范围的开始和结束地址。<字节值列表>指定一个或多个字节值的模式，或要搜索的字符串。用空格或逗号分隔每个字节值和下一个字节值。将字符串值包括在引号中。

17. 跟踪命令 T

T 命令用于单步跟踪指令的执行来调试程序，常被称为单步命令。T 命令有两种格式。

格式：

```
T [=地址]             ;逐条指令跟踪
T [=地址] [数值]       ;多条指令跟踪
```

T 命令从指定地址起执行一条或执行由[数值]参数指定条数的指令后停下来。每条指令执行后都要显示所有寄存器和标志位的值以及下一条指令。如未指定地址则从当前的 CS:IP 开始执行。注意：给出的执行地址前有一个等号，否则会被认为是被跟踪指令的条数(数值)。

T 命令逐条指令执行程序，遇到子程序(CALL)或中断调用(INT n)指令也不例外，会进入到子程序或中断服务程序当中执行。

18. 反汇编命令 U

在内存中，指令以二进制机器码的形式存放。U 命令用于将存储区的数据反汇编为汇编指令，有两种格式。

格式：

```
U [地址]                    ;从指定地址开始,反汇编 32 个字节 (80 列显示模式)
U 范围                      ;对指定范围的主存内容进行反汇编
```

如果不给出指定地址，则上一个 U 命令最后一条指令的下一个单元开始显示 32B。若还没有使用过 U 命令，则从当前 CS:IP 开始。

19. 写盘命令 W

格式：W <地址> <盘号> <起始逻辑扇区> <所读扇区个数 n>

该命令是将内存<地址>起始的一片单元内容写入指定扇区。当没有参数时，与 N 命令配合使用，完成文件写盘。

20. 显示命令列表?

格式：?

在 DEBUG 提示符下输入“?”命令，将显示所有命令的列表，起到联机帮助的作用。

附录 B

Masm for Windows 集成实验环境

Masm for Windows 集成实验环境是钟家民针对汇编语言初学者的特点开发的一个简单易用的汇编语言实验软件，支持 32 位与 64 位的 Windows 7/8/8.1 操作系统，支持 DOS 的 16/32 位汇编程序和 Windows 下的 32 位汇编程序(并提供调试通过的 35 个 Windows 汇编程序实例源代码)，它具有错误信息自动定位、关键字实时帮助并且在帮助中动画演示汇编指令的执行过程、语法着色、无限次撤销与恢复，WORD 式的查找、替换、定位，支持中文、长文件名等功能。

1. 软件安装与卸载

Masm for Windows 集成实验环境可安装在 Windows 2000、Windows XP、Windows Vista、Vista SP1、Windows 7 等操作系统。安装方法与其他 Windows 下的应用程序一样，双击安装文件开始安装，出现安装向导对话框，如图 B-1 所示。

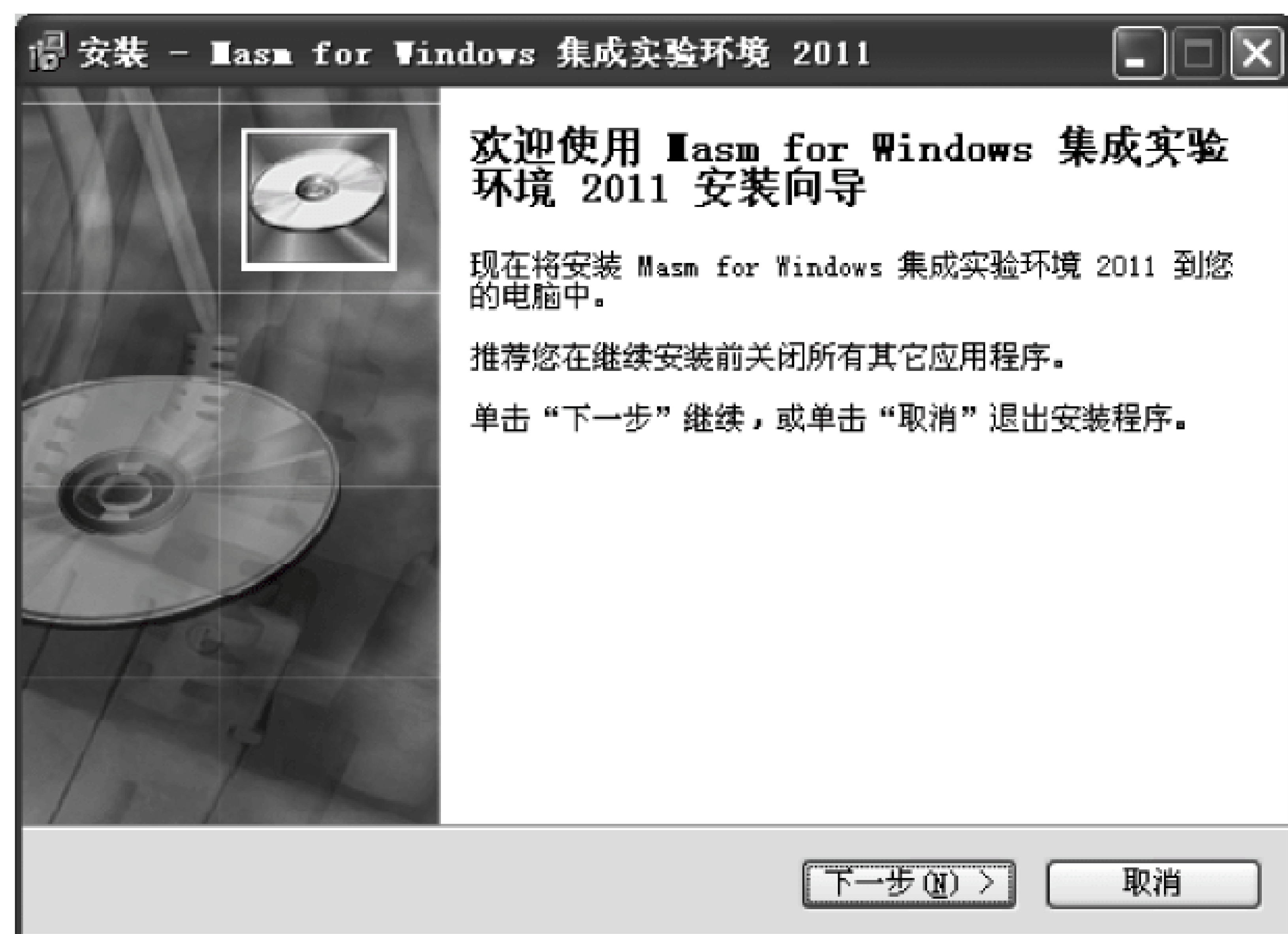


图 B-1　软件安装对话框

单击“下一步”按钮，按默认设置即可完成安装。对于免费安装版软件，安装成功后显示为共享版，能满足基本使用的要求，但有些联机帮助功能不能使用。如果要安装注册版，需要进行软件的注册验证。

卸载软件有两种方法：

(1) 在“控制面板”窗口中，双击“添加和删除程序”图标，然后在打开的“添加和删除程序”对话框中，选择“Masm for Windows 集成实验环境”进行删除即可。

(2) 选择“开始”|“程序”|“汇编语言集成实验环境”|“卸载 Masm for Windows 集成实验环境”，即可删除。

2. 软件的基本操作

运行“Masm for Windows 集成实验环境”后，主界面如图 B-2 所示。在程序书写区域，给出了完整段定义的汇编语言格式，可以直接利用该格式模板编写程序。当然，也可以不利用给出的格式模板，完全自主编写程序。

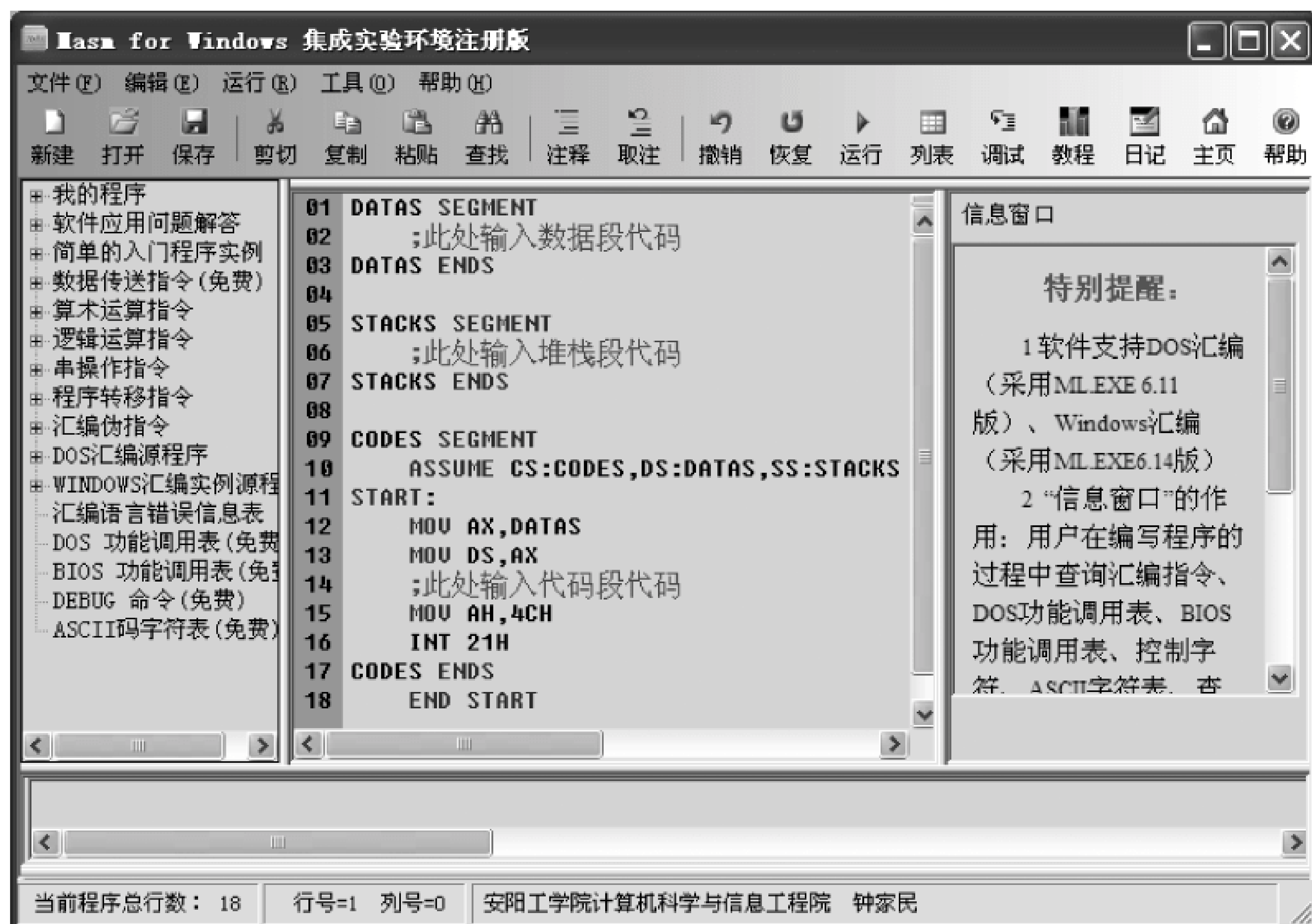


图 B-2　集成实验环境主界面

1）新建/打开程序

主界面默认状态是“新建”程序，如果是编辑已有的源文件，可以选择“文件”|“打开”或单击工具栏中的“打开”按钮，在出现的“打开”对话框中选择要打开的汇编语言程序文件。

2）编辑程序

在编写程序过程中，利用查找与替换功能修改变量。在程序输入区单击鼠标右键，在弹出的快捷菜单中选择“替换”，出现“查找和替换”对话框，在“查找内容”文本框中输入要查找的字，在“替换为”文本框中输入替换后的内容，如图 B-3 所示。



图 B-3　“查找和替换”对话框

在编写程序过程中，可以方便地给程序加注释。方法是：在要注释的程序行上单击，然后单击工具栏上的“注释”按钮。如果忘记某个指令的用法，只要用鼠标右击关键字，在弹出的快捷菜单中选择“实时帮助”，即可获得该指令的帮助。

编辑完成后，保存文件时选择“文件”|“保存”或单击工具栏中“保存”按钮，默认的扩展名为.ASM。

3）汇编、连接和运行

主界面的“运行”菜单如图 B-4 所示。选择“编译成目标文件”菜单项就可进行汇编，生

成目标文件.OBJ。当程序发生语法错误时,会自动定位程序中错误所在行的位置并高亮显示该行,修改好第一条发生错误行后,双击任一条错误信息,该软件定位程序中与之相对应错误所在行的位置并高亮显示该行,以便改正错误。

在“运行”菜单中,选择“生成可执行文件”菜单项,完成连接过程。如果连接过程报错,则需进行错误分析,然后重新调入编辑程序进行修改,然后重新汇编,再经过连接,直至无错误为止。连接以后,便产生了可执行程序文件(EXE 文件)。

实际上,汇编、连接、运行操作,已经集成在“运行”菜单中的“运行”菜单项或者工具栏中的“运行”按钮,可以直接单击菜单项或工具栏按钮,将汇编、连接、运行一体化。

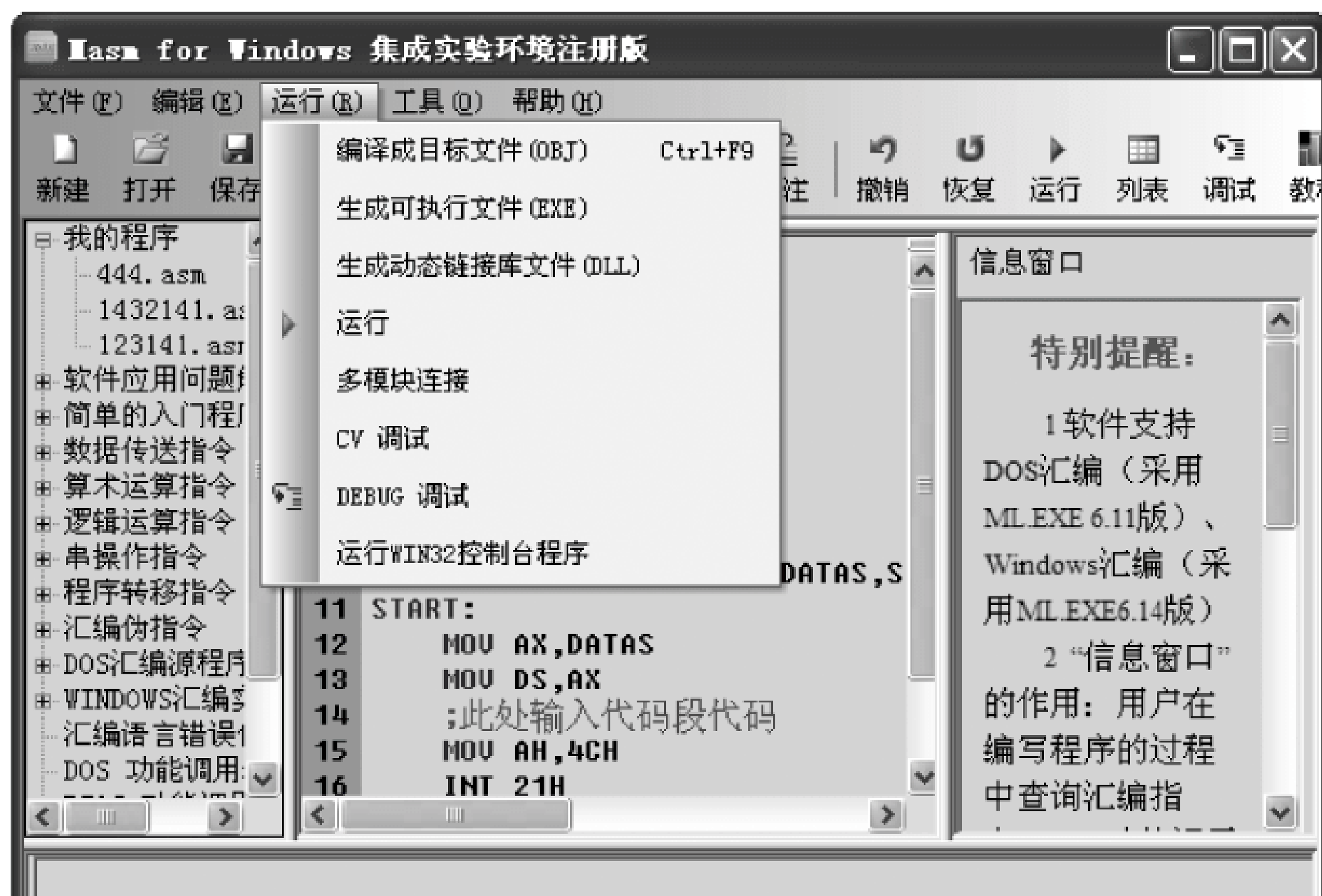


图 B-4 “运行”菜单的主要功能

4)动态调试

通过编辑、汇编和连接后的程序是可以执行的程序,可以在 DOS 下运行,也可以在集成实验环境中执行。在运行时发现逻辑错误如运行结果不正确,还需要进行动态调试,找到错误后,编辑修改程序,然后再重新进行汇编、连接、执行操作,直至没有错误时为止。

在 Masm for Windows 集成实验环境中提供了两种调试方法:CV 调试和 DEBUG 调试,可以根据用户的需要选择其中一种调试方法即可。集成实验环境中的 DEBUG 与 DOS 下的 DEBUG 操作方法相同,具体命令详见附录 A。

在 DEBUG 动态调试中,需要先生成 EXE 文件后,才能用 DEBUG 进行调试。DEBUG 的主要功能有显示和修改寄存器及内存单元的内容;按指定地址启动并运行程序;设置断点使程序分段运行,以便检查程序运行过程中的中间结果或确定程序出错的位置;反汇编被调试程序,它将一个可执行文件中的指令机器码反汇编成助记符指令并同时给出指令所在的内存地址;单条追踪或多条追踪被调试程序,它可以逐条指令执行或几条指令执行被调试程序,每执行一条(或几条)指令后,DEBUG 程序将中断程序的运行并提供有关结果信息等。

CV 调试就是利用 Code View Debugger 调试工具进行动态调试,该程序提供了多窗口全屏幕调试环境,窗口形式如图 B-5 所示。

CV 功能键如下:

F2:显示/隐含的寄存器组窗口;

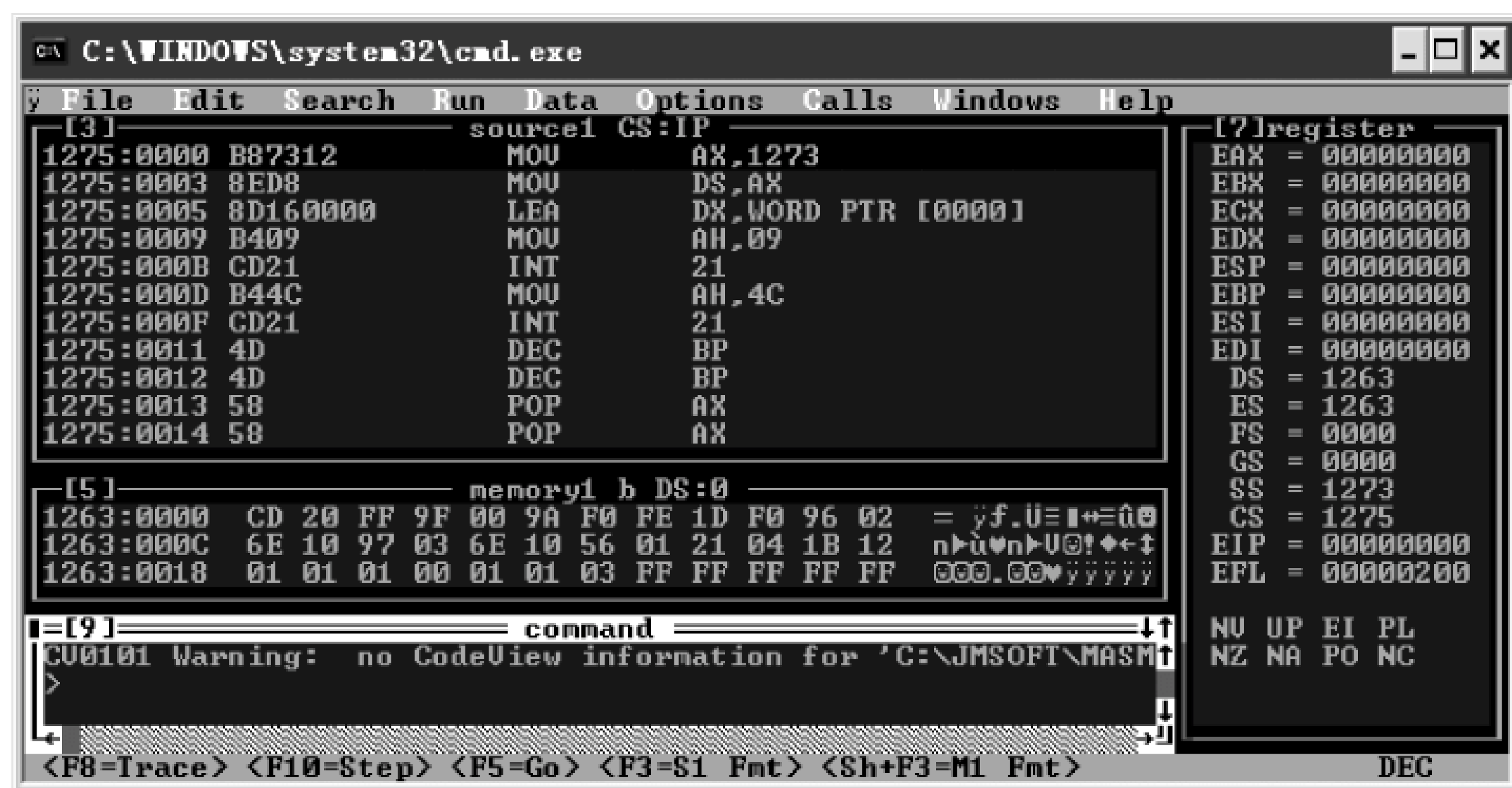


图 B-5　CV 调试窗口

F3：以不同的显示方式显示当前执行的程序；

F4：显示程序的输出屏幕；

F5/F7：执行到下一个逻辑断点，或到程序尾；

F6：依次进入当前屏幕所显示的窗口；

F8：单步执行指令，并进入被调用的子程序；

F9：在程序行中设置/取消断点，或用鼠标双击；

F10：单步执行指令，但不进入被调用的子程序。

该集成实验环境的默认调试工具是 CV 调试，如果要修改默认设置，可以选择“工具”|“选项”，在出现的“选项”对话框中进行设置，如图 B-6 所示。

在“选项”对话框中还可以管理自己的程序，将自己的程序保存在一个固定的文件夹内，便于查找。单击“我的程序文件夹”选项组后的“设置”按钮，在图 B-7 所示的对话框中，选择要设置的文件夹。这样，当本软件打开或保存程序时，软件会自动定位到所设置的文件夹，方便地打开或保存用户编写的汇编语言程序。



图 B-6　“选项”对话框

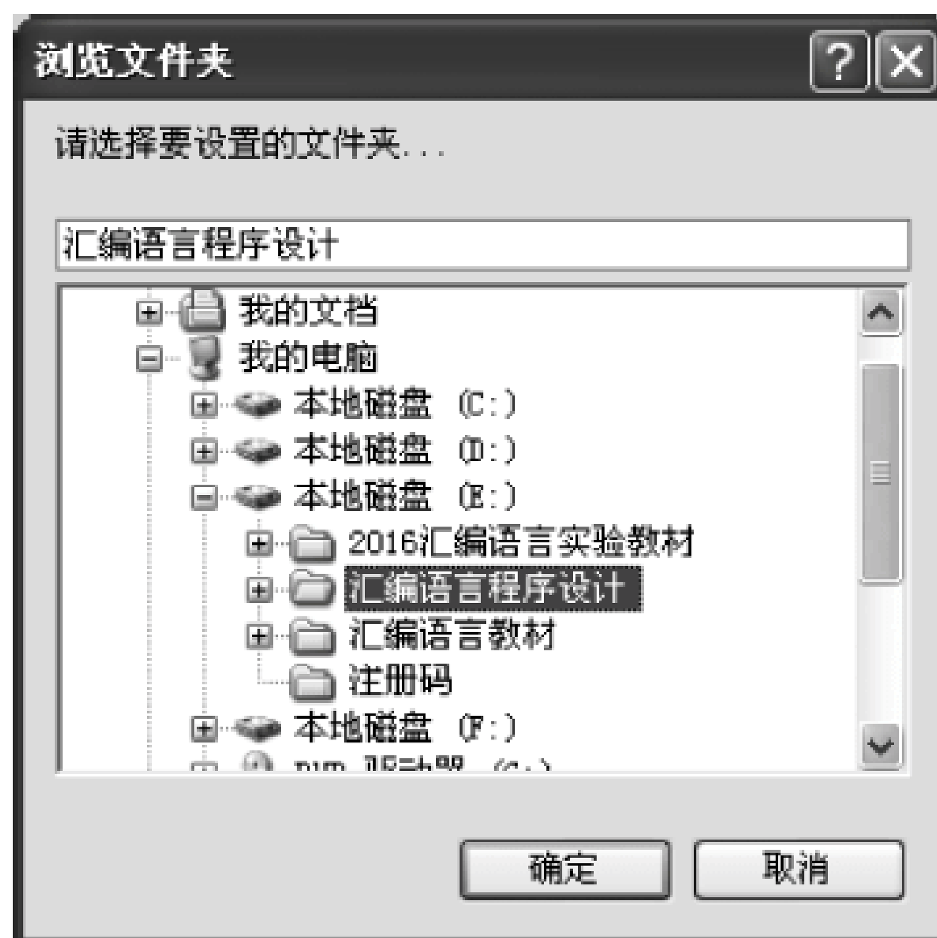


图 B-7　程序文件夹设置

关于该软件使用的其他问题，可参见系统自带的联机帮助，或登录家民软件专区网站：http://www.jiaminsoft.com/。

附录 C

ASCII 码表

二　进　制	十　进　制	十 六 进 制	缩写/字符	解　　释
00000000	0	00	NUL	空字符
00000001	1	01	SOH	标题开始
00000010	2	02	STX	正文开始
00000011	3	03	ETX	正文结束
00000100	4	04	EOT	传输结束
00000101	5	05	ENQ	请求
00000110	6	06	ACK	收到通知
00000111	7	07	BEL	响铃
00001000	8	08	BS	退格
00001001	9	09	HT	水平制表符
00001010	10	0A	LF	换行键
00001011	11	0B	VT	垂直制表符
00001100	12	0C	FF	换页键
00001101	13	0D	CR	回车键
00001110	14	0E	SO	不用切换
00001111	15	0F	SI	启用切换
00010000	16	10	DLE	数据链路转义
00010001	17	11	DC1	设备控制 1
00010010	18	12	DC2	设备控制 2
00010011	19	13	DC3	设备控制 3
00010100	20	14	DC4	设备控制 4
00010101	21	15	NAK	拒绝接收
00010110	22	16	SYN	同步空闲
00010111	23	17	ETB	结束传输块
00011000	24	18	CAN	取消
00011001	25	19	EM	媒介结束
00011010	26	1A	SUB	代替
00011011	27	1B	ESC	换码(溢出)
00011100	28	1C	FS	文件分隔符
00011101	29	1D	GS	分组符
00011110	30	1E	RS	记录分隔符
00011111	31	1F	US	单元分隔符

续表

二　进　制	十　进　制	十六进制	缩写/字符	解　　释
00100000	32	20	(Space)	空格
00100001	33	21	!	叹号
00100010	34	22	"	双引号
00100011	35	23	#	井号
00100100	36	24	$	美元符
00100101	37	25	%	百分号
00100110	38	26	&	和号
00100111	39	27	'	闭单引号
00101000	40	28	(	开括号
00101001	41	29	)	闭括号
00101010	42	2A	*	星号
00101011	43	2B	+	加号
00101100	44	2C	,	逗号
00101101	45	2D	-	减号/破折号
00101110	46	2E	.	句号
00101111	47	2F	/	斜杠
00110000	48	30	0	数字 0
00110001	49	31	1	数字 1
00110010	50	32	2	数字 2
00110011	51	33	3	数字 3
00110100	52	34	4	数字 4
00110101	53	35	5	数字 5
00110110	54	36	6	数字 6
00110111	55	37	7	数字 7
00111000	56	38	8	数字 8
00111001	57	39	9	数字 9
00111010	58	3A	:	冒号
00111011	59	3B	;	分号
00111100	60	3C	<	小于
00111101	61	3D	=	等号
00111110	62	3E	>	大于
00111111	63	3F	?	问号
01000000	64	40	@	电子邮件符号
01000001	65	41	A	大写字母 A
01000010	66	42	B	大写字母 B
01000011	67	43	C	大写字母 C
01000100	68	44	D	大写字母 D
01000101	69	45	E	大写字母 E
01000110	70	46	F	大写字母 F
01000111	71	47	G	大写字母 G
01001000	72	48	H	大写字母 H
01001001	73	49	I	大写字母 I

续表

二 进 制	十 进 制	十 六 进 制	缩写/字符	解 释
01001010	74	4A	J	大写字母 J
01001011	75	4B	K	大写字母 K
01001100	76	4C	L	大写字母 L
01001101	77	4D	M	大写字母 M
01001110	78	4E	N	大写字母 N
01001111	79	4F	O	大写字母 O
01010000	80	50	P	大写字母 P
01010001	81	51	Q	大写字母 Q
01010010	82	52	R	大写字母 R
01010011	83	53	S	大写字母 S
01010100	84	54	T	大写字母 T
01010101	85	55	U	大写字母 U
01010110	86	56	V	大写字母 V
01010111	87	57	W	大写字母 W
01011000	88	58	X	大写字母 X
01011001	89	59	Y	大写字母 Y
01011010	90	5A	Z	大写字母 Z
01011011	91	5B	[	开方括号
01011100	92	5C	\	反斜杠
01011101	93	5D	]	闭方括号
01011110	94	5E	^	脱字符
01011111	95	5F	_	下画线
01100000	96	60	`	开单引号
01100001	97	61	a	小写字母 a
01100010	98	62	b	小写字母 b
01100011	99	63	c	小写字母 c
01100100	100	64	d	小写字母 d
01100101	101	65	e	小写字母 e
01100110	102	66	f	小写字母 f
01100111	103	67	g	小写字母 g
01101000	104	68	h	小写字母 h
01101001	105	69	i	小写字母 i
01101010	106	6A	j	小写字母 j
01101011	107	6B	k	小写字母 k
01101100	108	6C	l	小写字母 l
01101101	109	6D	m	小写字母 m
01101110	110	6E	n	小写字母 n
01101111	111	6F	o	小写字母 o
01110000	112	70	p	小写字母 p
01110001	113	71	q	小写字母 q
01110010	114	72	r	小写字母 r
01110011	115	73	s	小写字母 s

续表

二　进　制	十　进　制	十六进制	缩写/字符	解　　释
01110100	116	74	t	小写字母 t
01110101	117	75	u	小写字母 u
01110110	118	76	v	小写字母 v
01110111	119	77	w	小写字母 w
01111000	120	78	x	小写字母 x
01111001	121	79	y	小写字母 y
01111010	122	7A	z	小写字母 z
01111011	123	7B	{	开花括号
01111100	124	7C	|	垂线
01111101	125	7D	}	闭花括号
01111110	126	7E	～	波浪号
01111111	127	7F	DEL	删除

附录 D

DOS 系统功能调用

功能号 AH	功　　能	入 口 参 数	出 口 参 数
00	程序终止(同 INT 20H)	CS=程序段前缀	
01	键盘输入并回显		AL=输入字符
02	显示输出	DL=输出字符	
03	异步通信输入		AL=输入数据
04	异步通信输出	DL=输出数据	
05	打印机输出	DL=输出字符	
06	直接控制台 I/O	DL=FF(输入) DL=字符(输出)	AL=输入字符
07	键盘输入(无回显)		AL=输入字符
08	键盘输入(无回显) 检测 Ctrl+Break		AL=输入字符
09	显示字符串	DS:DX=串地址 '$'结束字符串	
0A	键盘输入到缓冲区	DS:DX=缓冲区首地址 (DS:DX)=缓冲区最大字符数	(DS:DX+1)=实际输入的字符数
0B	检验键盘状态		AL=00 有输入 AL=FF 无输入
0C	清除输入缓冲区并请求指定的输入功能	AL=输入功能号 (1,6,7,8,A)	
0D	磁盘复位		清除文件缓冲区
0E	指定当前缺省的磁盘驱动器	DL=驱动器号 0=A,1=B,…	AL=驱动器数
0F	打开文件	DS:DX=FCB 首地址	AL=00 文件找到 AL=FF 文件未找到
10	关闭文件	DS:DX=FCB 首地址	AL=00 目录修改成功 AL=FF 目录中未找到文件
11	查找第一个目录项	DS:DX=FCB 首地址	AL=00 找到 AL=FF 未找到
12	查找下一个目录项	DS:DX=FCB 首地址 (文件中带有 * 或?)	AL=00 找到 AL=FF 未找到

续表

功能号 AH	功　能	入口参数	出口参数
13	删除文件	DS:DX=FCB 首地址	AL=00 删除成功 AL=FF 未找到
14	顺序读	DS:DX=FCB 首地址	AL=00 读成功 AL=01 文件结束，记录中无数据 AL =02 DTA 空间不够 AL=03 文件结束，记录不完整
15	顺序写	DS:DX=FCB 首地址	AL=00 写成功 AL =01 盘满 AL =02 DTA 空间不够
16	建文件	DS:DX=FCB 首地址	AL=00 建立成功 AL=FF 无磁盘空间
17	文件改名	DS:DX=FCB 首地址 (DS:DX+1)=旧文件名 (DS:DX+17)=新文件名	AL=00 成功 AL=FF 未成功
19	取当前默认磁盘驱动器		AL=默认的驱动器号 0=A,1=B,2=C,…
1A	置 DTA 地址	DS:DX=DTA 地址	
1B	取默认驱动器 FAT 信息		AL=每簇的扇区数 DS:BX=FAT 标识字节 CX=物理扇区大小 DX=默认驱动器的簇数
1C	取任一驱动器 FAT 信息	DL=驱动器号	AL=每簇的扇区数 DS:BX=FAT 标识字节 CX=物理扇区大小 DX=默认驱动器的簇数
1D	DOS 内部使用		
1E	DOS 内部使用		
1F	DOS 内部使用		
20	DOS 内部使用		
21	随机读	DS:DX=FCB 首地址	AL=00 读成功 AL=01 文件结束 AL=02 缓冲区溢出 AL=03 缓冲区不满
22	随机写	DS:DX=FCB 首地址	AL=00 写成功 AL=01 盘满 AL =02 缓冲区溢出
23	测定文件大小	DS:DX=FCB 首地址	AL=00 成功(文件长度填入FCB) AL=FF 未找到
24	设置随机记录号	DS:DX=FCB 首地址	

续表

功能号 AH	功　　能	入 口 参 数	出 口 参 数
25	设置中断向量	DS:DX＝中断向量 AL＝中断类型号	
26	建立程序段前缀	DX＝新的程序段前缀	
27	随机分块读	DS:DX＝FCB 首地址 CX＝记录数	AL＝00 读成功 AL＝01 文件结束 AL＝02 缓冲区太小，传输结束 AL ＝03 缓冲区不满
28	随机分块写	DS:DX＝FCB 首地址 CX＝记录数	AL＝00 写成功 AL ＝01 盘满 AL ＝02 缓冲区溢出
29	分析文件名	ES:DI＝FCB 首地址 DS:SI＝ASCIIZ 串 AL＝控制分析标志	AL＝00 标准文件 AL＝01 多义文件 AL＝02 非法盘符
2A	取日期		CX＝年 DH:DL＝月:日(二进制)
2B	设置日期	CX:DH:DL＝年:月:日	AL＝00 成功 AL＝FF 无效
2C	取时间		CH:CL＝时:分 DH:DL＝秒:1/100 秒
2D	设置时间	CH:CL＝时:分 DH:DL＝秒:1/100 秒	AL＝00 成功 AL＝FF 无效
2E	置磁盘自动读写标志	AL＝00 关闭标志 AL＝01 打开标志	
2F	取磁盘缓冲区的首址		ES:BX＝缓冲区首址
30	取 DOS 版本号		AH＝发行号，AL＝版本
31	结束并驻留	AL＝返回码 DX＝驻留区大小	
32	取驱动器参数块	DL＝驱动器号	AL＝FFH 驱动器无效 DS:BX＝驱动器参数块地址
33	Ctrl＋Break 检测	AL＝00 取状态 AL＝01 置状态(DL) DL＝00 关闭检测 DL＝01 打开检测	DL＝00 关闭 Ctrl＋Break 检测 DL＝01 打开 Ctrl＋Break 检测
35	取中断向量	AL＝中断类型	ES:BX＝中断向量
36	取空闲磁盘空间	DL＝驱动器号 0＝默认，1＝A，2＝B，…	成功：AX＝每簇扇区数 　BX＝有效簇数 　CX＝每扇区字节数 　DX＝总簇数 失败：AX＝FFFF
37	DOS 内部使用		
38	置/取国家信息	DS:DX＝信息区首地址	BX＝国家码(国际电话前缀码) AX＝错误码

续表

功能号 AH	功　　能	入 口 参 数	出 口 参 数
39	建立子目录(MKDIR)	DS:DX=ASCII 字符串地址	AX=错误码
3A	删除子目录(RMDIR)	DS:DX=ASCII 字符串地址	AX=错误码
3B	改变当前目录(CHDIR)	DS:DX=ASCII 字符串地址	AX=错误码
3C	建立文件	DS:DX=ASCII 字符串地址 CX=文件属性	成功：AX=文件代号 错误：AX=错误码
3D	打开文件	DS:DX=ASCII 字符串地址 AL=0 读 AL=1 写 AL=2 读/写	成功：AX=文件代号 错误：AX=错误码
3E	关闭文件	BX=文件代号	失败:AX=错误码
3F	读文件或设备	DS:DX=数据缓冲区地址 BX=文件代号 CX=读取的字节数	读成功：AX=实际读入的字节数 AX=0 已到文件尾 读出错：AX=错误码
40	写文件或设备	DS:DX=数据缓冲区地址 BX=文件代号 CX=写入的字节数	写成功：AX=实际写入的字节数 写出错：AX=错误码
41	删除文件	DS:DX=ASCII 字符串地址	成功：AX=00 出错：AX=错误码(2,5)
42	移动文件指针	BX=文件代号 CX:DX=位移量 AL=移动方式(0:从文件头绝对位移,1:从当前位置相对移动,2:从文件尾绝对位移)	成功：DX:AX=新文件指针位置 出错：AX=错误码
43	置/取文件属性	DS:DX=ASCIIZ 串地址 AL=0 取文件属性 AL=1 置文件属性 CX=文件属性	成功：CX=文件属性 失败：CX=错误码
44	设备文件 I/O 控制	BX=文件代号 AL=0 取状态 AL=1 置状态 DX AL=2 读数据 AL=3 写数据 AL=4 同 2,但由 BL 指出驱动器号 AL=5 同 3,但由 BL 指出驱动器号 AL=6 取输入状态 AL=7 取输出状态	DX=设备信息
45	复制文件代号	BX=文件代号 1	成功：AX=文件代号 2 失败：AX=错误码

续表

功能号 AH	功　　能	入 口 参 数	出 口 参 数
46	人工复制文件代号	BX=文件代号 1 CX=文件代号 2	失败：AX=错误码
47	取当前目录路径名	DL=驱动器号 DS:SI=ASCII 字符串首地址	成功：DS:SI=ASCII 字符串首地址 失败：AX=出错码
48	分配内存空间	BX=申请内存容量	成功：AX=分配内存首地址 失败：BX=最大可用内存
49	释放内容空间	ES=内存起始段地址	失败：AX=错误码
4A	调整已分配的存储块	ES=原内存起始地址 BX=再申请的容量	失败：BX=最大可用空间 AX=错误码
4B	装配/执行程序	DS:DX = ASCII 字符串首地址 ES:BX=参数区首地址 AL=0 装入执行 AL=3 装入不执行	失败：AX=错误码
4C	带返回码结束	AL=返回码	
4D	取返回代码		AX=返回代码
4E	查找第一个匹配文件	DS:DX = ASCII 字符串首地址 CX=属性	AX=出错代码(02,18)
4F	查找下一个匹配文件	DS:DX = ASCII 字符串首地址 (文件名中带有？或＊)	AX=出错代码(18)
50	置 PSP 段地址	BX=新 PSP 段地址	
51	取 PSP 段地址		BX=当前 PSP 段地址
52	取磁盘参数块		ES:BX=参数块链表指针
53	BIOS 参数块转换为 DOS 参数块	DS:SI=BPB 的指针 ES:BP=DPB 的指针	
54	取盘自动读写标志		AL=当前标志值
55	建立 PSP	DX=建立 PSP 段地址	
56	文件改名	DS:DX=ASCII 字符串(旧) ES:DI=ASCII 字符串(新)	AX=出错码(03,05,17)
57	置/取文件日期和时间	BX=文件代号 AL=0 读取 AL=1 设置(DX:CX)	DX:CX=日期和时间 失败：AX=错误码
58	取/置分配策略码	AL=0 取码 AL=1 置码(BX)	成功：AX=策略码 失败：AX=错误码
59	取扩充错误码		AX=扩充错误码 BH=错误类型 BL=建议的操作 CH=错误场所

续表

功能号 AH	功　　能	入口参数	出口参数
5A	建立临时文件	CX＝文件属性 DS:DX＝ASCII 字符串地址	成功：AX＝文件代号 失败：AX＝错误码
5B	建立新文件	CX＝文件属性 DS:DX＝ASCIIZ 串地址	成功：AX＝文件代号 失败：AX＝错误码
5C	控制文件存取	AL＝00 封锁 AL＝01 开启 BX＝文件代号 CX:DX＝文件位移 SI:DI＝文件长度	失败：AX＝错误码
5D	取/置严重错误标志地址	AL＝06 取错误标志地址 AL＝0A 置 ERROR 结构指针	DS:SI＝严重错误标志地址
5E	DOS 内部使用		
5F	DOS 内部使用		
60	扩展为全路径名	DS:SI＝字符串地址 ES:DI＝工作缓冲区地址	失败：AX＝错误代码
61	DOS 内部使用		
62	取程序段前缀		BX＝PSP 地址
63	DOS 内部使用		
64	DOS 内部使用		
65	DOS 内部使用		
66	DOS 内部使用		
67	DOS 内部使用		
68	刷新缓冲区数据到磁盘	AL＝文件代号	失败：AX＝错误代码
69	DOS 内部使用		
6A	DOS 内部使用		
6B	DOS 内部使用		
6C	扩充的文件打开/建立	AL＝访问权限 BX＝打开方式 CX＝文件属性 DS:SI＝ASCII 字符串地址	成功：AX＝文件代号 　　　CX＝采取的动作 失败：AX＝错误代码

附录 E

模拟试题及参考答案

模拟试题一

一、单选题(15 分)

1．CPU 要访问的某一存储单元的实际地址称为(　　)。

A．物理地址　　B．逻辑地址　　C．段地址　　D．有效地址

2．下面各指令中，正确的是(　　)。

A．MOV　AH，10[BX+BP]　　B．MOV [DI]，[SI]

C．MOV WORD PTR [BX]，0100H　　D．ADC　AH，DX

3．下列指令属于寄存器寻址的是(　　)。

A．MOV　ES，AX　　B．MOV　DX，DS：[BP][SI]

C．MOV　CX，COUNT[BX]　　D．MOV　[DI]，BL

4．下面的解释正确的是(　　)。

A．指令“XOR AX，AX”执行后，AX 内容不变，但设置了标志位

B．指令“OR DX，1000H”执行后，将 DX 最高位置 1，其余各位置 0

C．指令“NOT AX”执行后，将 AX 清 0

D．指令“AND AX，0FH”执行后，分离出 AL 低四位

5．若定义“BUF　DW　1，2，30 DUP(5)”，则为变量 BUF 分配的内存单元数是(　　)字节。

A．32　　B．64　　C．30　　D．35

6．要将两个字符 A、B 顺序存放在连续两个字节存储单元中，可选用的数据定义语句是(　　)。

A．BUF1　DB 'AB'　　B．BUF1　DW 'AB'

C．BUF1　DB　0AH，0BH　　D．BUF1　DW　0B0AH

7．它通常用来存放代码段中的偏移地址，在程序运行过程中，它始终指向下一条指令的首地址，这个寄存器是(　　)。

A．SP　　B．BP　　C．IP　　D．CS

8．在默认情况下，指令“MOV　[BP+SI]，AL”中的目的操作数使用(　　)段寄存器。

A．CS　　B．DS　　C．SS　　D．ES

9．在下列指令中，操作数在代码段的是(　　)。

A．ADD AL，DS：[2000H]　　B．MOV [BX+SI+100]，AX

C. CMP AL,CL　　D. MOV AL,32H

10. 一般在循环指令中作为循环次数的寄存器是(　　)。

A. AX　B. BX　C. CX　D. DX

11. 指令“MOV AL,[2000H]”中,源操作数的物理地址为(　　)。

A. CS * 16+2000H　　B. DS * 16+2000H

C. SS * 16+2000H　　D. 2000H

12. 无论BH中原有的数是奇数或偶数,若要使BH中的数一定为奇数,应执行的指令是(　　)。

A. ADD BH,01H　　B. SUB BH,01H

C. XOR BH,01H　　D. OR BH,01H

13. 设AL=0AH,下列指令执行后能使AL=05H的是(　　)。

A. NOT AL　　B. AND AL,0FH

C. XOR AL,0FH　　D. OR AL,0FH

14. 下列指令执行后,标志位CF一定为0的是(　　)。

A. SBB BX,BX　B. DEC AL　C. MOV CX,0　D. SUB AX,AX

15. 执行下列指令后,AX寄存器的内容是(　　)。

```
DA1   DW   1122H,4433H,5678H
      ⋮
 MOV AX,DA1 + 3
```

A. 7844H　B. 3356H　C. 5633H　D. 5678H

二、填空题(16分)

1. 若DS=3000H,SI=2000H,COUNT=58H,指令“MOV AX,COUNT [SI]”中,源操作数的有效地址EA为(__1__),其物理地址为(__2__)。

2. 在8088/8086微处理器中,存储器是分段组织的。有四个段寄存器专门用来存放段地址,它们分别是代码段寄存器CS、数据段寄存器DS、(__3__)和(__4__)。

3. 在8086CPU寄存器中,AX、(__5__)、(__6__)、(__7__)四个寄存器,既可以作为16位寄存器来使用,也可以作为两个8位寄存器来使用,其中(__8__)寄存器可以在寄存器间接寻址中用做地址指针。

4. 存储器由许多存储单元组成,存储单元是以(__9__)为单位进行组织。

5. 若累加器AX中的内容为7865H,则执行指令“SUB AX,4041H”后,AX中的内容为(__10__);若再接着执行“CMP AX,4041H”后,标志位CF为(__11__)。

6. 设BX=6F30H,BP=0200H,SI=0046H,SS=2F00H,(2F246H)=4154H,执行“XCHG BX,[BP+SI]”后,BX中的内容为(__12__)。

7. 若当前(AL)=38H,标志位CF=0,则执行“ADC AL,47H”后,AL中的内容为(__13__);若再接着执行一条DAA指令,则AL中的内容为(__14__)。

8. 若(SP)=1362H,(AX)=3045H,(BX)=3516H,则执行PUSH AX后,SP的内容为(__15__);若再接着执行PUSH BX及POP DX两条指令后,DX的内容为(__16__)。

三、简答题(14分)

1. (8分)各用一条指令分别实现以下功能。

(1) 寄存器 AX 和 DX 的内容相加,结果存入 AX 寄存器中。

(2) 将寄存器 BX 的内容传送至 CX 寄存器中。

(3) 若运算结果为 0 则转移至 NEXT 标号处执行。

(4) 将 AX 中的无符号数乘以 2。

2. (4 分)写出完成下列操作的伪指令语句。

(1) 将字节数据 34H、79H 存放在字节变量 DA1 为首地址的存储单元中。

(2) 将字数据 6728H、5A34H 存放在字节变量 DA2 为首地址的存储单元中,而且不可改变字数据高低字节存放的约定。

3. (2 分)下面指令有什么语法错误? 如何改正?

```
MOV AX,DL
```

四、程序分析(20 分)

1. (6 分)现有 DS=2000H,BX=0100H,SI=0002H,(20100H)=12H,(20101H)=34H,(20102H)=56H,(20103H)=78H,(21200H)=2AH,(21201H)=4CH,(21202H)=0B7H,(21203H)=65H,试说明下列各条指令的寻址方式及执行后 AX 寄存器的内容。

(1) MOX AX,BX

(2) MOV AX,[1200H]

(3) MOV AX,[BX]

2. (2 分)执行下列指令后,(BX)=?

```
MOV   AX,5432H
MOV   BX,AX
```

3. (2 分)执行下列指令后,(AX)=?

```
MOV   AX,3768H
MOV   BX,AX
MOV   AH,AL
```

4. (2 分)
```
MOV   AL,35H
MOV   BL,39H
ADD   AL,BL
```

执行后(AL)=? 状态标志位 CF=?

5. (2 分)执行下列指令后,(AL) = ?

```
BUF DW 2152H,3416H,5731H,4684H
         ⋮
   LEA BX,BUF
   MOV AL,2
   XLAT
```

6. (2 分)阅读下列程序段,如果在程序段运行期间,当执行完指令 INC BX 后(BX)=3 时,此时寄存器 CX=? AL=?

```
DATA1    DB 10 DUP(2)
DATA2    DW 10 DUP(0301H)
```

```
        ⋮
    MOV   BX,0
    MOV   AL,0
    MOV   CX,10
NEXT: ADD AL, DATA1 [BX]
    ADD AL,BYTE PTR DATA2[BX]
    INC BX
    LOOP NEXT
```

7. (4 分)

```
DA1   DB   'ABCDEFGHMN'
DA2   DB   10   DUP(?)
         ⋮
       MOV   CX,3
       MOV   BX,3
       MOV   DI,0
 LOPN: MOV   AL,DA1[BX]
       MOV   [DI + DA2],AL
       INC    BX
       INC    DI
       LOOP   LOPN
```

(1) 写出以 DA1 为首地址的 10 个字节单元的内容(可画示意图)。

(2) 程序执行完后,写出以 DA2 为首地址的前三个字节单元的内容(可画示意图)。

五、程序填空(15 分)

1. (6 分)假设从键盘上输入一个四位十进制数,要求将它以非压缩 BCD 码的形式存储在 BUF 开始的内存缓冲区中。

```
DSEG   SEGMENT                ;数据段
  BUF   DB   4 DUP(?)
DSEG   ENDS
CODE   SEGMENT                ;代码段
   ASSUME CS:CODE,DS:DSEG
START: MOV AX,DSEG
       MOV DS,AX
       MOV CX,4                ;CX 为计数器
       MOV DI,OFFSET BUF       ;设置地址指针
       MOV AH,01H
NEXT:  INT 21H
       ______(1)______         ;ASCII 码转换为非压缩 BCD
       MOV [DI],AL
       ______(2)______
       LOOP NEXT
       MOV   AH,4CH
       INT   21H
CODE   ENDS
       END   START
```

2. (9分)对BLOCK开始存储的一组字节无符号数进行比较,找出最大数,并将其存储在MAX单元中。程序实现方法:首先,将第一个数送寄存器AL;然后,AL依次与下一个数比较,并保留较大的数;最后,AL中的内容即为最大数。在下列程序中,用CX做计数器,SI做地址指针,采用寄存器间接寻址。

```
BLOCK   DB 05H,12H,3BH,43H,60H,0CH
        DB 8AH,0ABH,37H,0FFH,26H,37H
COUNT   EQU   $ - OFFSET   BLOCK
MAX     DB   ?
              ⋮
        MOV SI,OFFSET BLOCK
        MOV CX,COUNT
        MOV AL,[SI]
        INC SI
        DEC CX
LKF :   ______(1)______        ;比较
        JA NEXT                ;若(AL)大于下一个数,则转至NEXT
        ______(2)______
NEXT:   INC SI
        LOOP   LKF
        ______(3)______        ;保存最大数
        ⋮
```

六、程序设计(20分)

1. (10分)先显示输出字符串"Please input N:",然后从键盘上输入任意一个3~9的数字N,则在显示器上输出一个N个*号组成的图形。例如:输入4,则输出****。

2. (10分)设某班学生人数为40人,定义一个数据区,存放该班学生某门课程的百分制成绩,编写程序求此课程的全班总分并存于随后的存储单元(学生成绩可自行拟定)。

模拟试题二

一、单选题(15分)

1. 在8086指令系统中,不可用来访问存储器操作数的是(　　)。

A. 直接寻址方式　　B. 寄存器间接寻址方式

C. 寄存器寻址方式　　D. 寄存器相对寻址方式

2. 下面各指令中,错误的是(　　)。

A. MOV　AH,10[BX][DX]　　B. MOV DI,SI

C. MOV　[BX],DX　　D. ADC　AH,DL

3. 下列指令属于寄存器间接寻址的是(　　)。

A. MOV　ES,AX　　B. MOV　DX,DS:[BP][SI]

C. MOV　CX,COUNT[BX]　　D. MOV　[DI],BL

4. 实现累加器AX清零的指令是(　　)。

A. XOR AX,AX　　B. OR AX,AX

C. NOT AX　　D. AND AX,AX

5. 若定义“BUF　DW　1,2,3”,则为变量 BUF 分配的内存单元数是(　　)字节。

A. 3　　B. 4　　C. 5　　D. 6

6. 以 BUF1 为首地址的数据区中顺序存放数据:"A","B",0,"C","D"。可选用的数据定义语句是(　　)。

A. BUF1　DB 'AB','CD'　　B. BUF1　DW 'AB','CD'

C. BUF1　DB　'AB',0,'CD'　　D. BUF1　DW 'AB',0,'CD'

7. 它通常用来指向堆栈段栈顶存储单元的偏移量,在堆栈操作过程中,随着堆栈中数据的变化,其值也做相应的变化,这个寄存器是(　　)。

A. SS　　B. BP　　C. IP　　D. SP

8. 设 DS=27FCH,有一个数据存储单元的偏移地址为 8640H,该数据存储单元的物理地址是(　　)。

A. 30500H　　B. 30600H　　C. AE3CH　　D. 0AE3CH

9. 在下列指令的寻址方式中,操作数直接存储在指令中,紧跟在指令操作码背后的是(　　)。

A. ADD AL,[2000H]　　B. MOV [BX+SI+100],AX

C. MOV AL,0AH　　D. AND AL,AH

10. 重复串操作前缀 REP 可与 MOVS、LODS、STOS 指令配合使用,使 REP 前缀后面的串操作指令重复执行,重复的次数要事先存放在寄存器(　　)中。

A. AX　　B. BX　　C. CX　　D. DX

11. 在下列伪指令中,指示汇编程序已定义的段与段寄存器之间的对应关系(即用于指定某逻辑段应通过哪个段寄存器寻址)的是(　　)。

A. ORG　　B. ASSUME　　C. SEGMENT　　D. EQU

12. 若执行“SBB AL,DH”指令后,使得标志位 ZF=1,则表明 AL 寄存器(　　)。

A. 结果为零　　B. 结果为负　　C. 结果为正　　D. 结果不为零

13. 设 AL=35H,下列指令执行后能使 AL=05H 的是(　　)。

A. SUB AL,30　　B. AND AL,0FH

C. XOR AL,0FH　　D. OR AL,30H

14. 下列指令中,操作数使用正确的是(　　)。

A. MOV　CH,10[BX]　　B. MUL　AL,BL

C. SHR　BX,4　　D. ADC　AH,DX

15. 在程序运行过程中,确定下一条指令的物理地址的计算表达式是(　　)。

A. DS*16+DI　　B. CS*16+IP　　C. SS*16+SP　　D. ES*16+SI

二、填空题(16 分)

1. 有一个 32KB 的存储区,首地址为 2345H:0001H,则其首单元的物理地址为(__1__),末单元的物理地址为(__2__)。

2. 在 8088/8086 微处理器中,存储器可以划分为若干逻辑段,每段最大是(__3__)字节单元,每段最少是(__4__)字节单元。

3. 在 DOS 功能调用中,通常功能号放在寄存器(__5__)中。对于 1 号 DOS 功能调用,通过键盘输入一个字符,则该字符的 ASCII 码存入寄存器(__6__)中。对于 2 号 DOS 功能

调用,待输出的字符的 ASCII 码要放在寄存器(___7___)中。

4. 指令“MOV BX,2017H”执行后,寄存器 BL 的内容是(___8___),BH 的内容是(___9___)。

5. 若累加器 AX 中的内容为 7865H,则执行指令“XOR AX,AX”后,标志位 CF 为(___10___);若再接着执行“ADD AX,9000H”后,则 AX 的内容为(___11___)。

6. 设 BX=6F76H,BP=0246H,SS=2F00H,(2F246H)=4192H,则执行交换指令“XCHG BX,[BP]”后,BX 中的内容为(___12___)。

7. 若当前(AL)=38H,(DS)=1000H,(BX)=2000H,(12000H)=56H,则执行“ADD AL,[BX]”后,AL 中的内容为(___13___);若再接着执行一条 DAA 指令,则 AL 中的内容为(___14___)。

8. 若(SP)=1728H,(AX)=6189H,(BX)=4325H,则执行 PUSH AX 及 PUSH BX 两条指令后,SP 的内容为(___15___);若再接着执行一条 POP AX 指令后,AX 的内容为(___16___)。

三、简答题(14 分)

1. (6 分)分别写出实现以下功能的汇编指令。

(1) 寄存器 AX 的内容减去 1024H 后,结果存入 AX 寄存器中,不考虑借位。

(2) 将 BH 寄存器中的数据传送至 DL 寄存器中。

(3) 若运算结果为负,则转移至 NEXT 标号处执行。

2. (4 分)假设数据段中的数据定义如下:

```
BUF1   DB   100 DUP(5)
BUF2   DW   1234H,6789H,50C8H,423AH
```

(1) 将 BUF1 的偏移地址存入 BX 寄存器;

(2) 将 BUF2 的第三个字数据(50C8H)送入 CX 寄存器(要求:不能采用立即寻址方式实现)。

3. (4 分)用 DW 语句改写下述用 DB 定义的两个语句,使它们在存储单元中的数据存储顺序完全相同。

```
DA1   DB 'ABCD'
DA2   DB 12H,34H,56H,78H
```

四、程序分析(20 分)

1. (6 分)说明下列指令的寻址方式并写出存储器操作数的物理地址表达式。

(1) `ADC   AX,[SI+5]`

(2) `MOV   [BP],BX`

(3) `MOV   AX,[BX][SI]`

2. (2 分)
```
MOV   AL,32H
MOV   AH,79H
MOV   BX,AX
```

执行后(BX)=?

3. (2 分)执行下列指令后,(AX)=?

```
MOV   BX,3368H
```

```
MOV   AX,BX
INC   AX
OR   AX,BX
```

4. (2 分)执行下列指令后,(AL)=? 状态标志位 CF=?

```
MOV AL,255
ADD AL,1
```

5. (2 分)执行下列指令后,(AX) = ?

```
     ⋮
TAB1    DW   1167H,1284H,2917H
     ⋮
    MOV   BX,OFFSET TAB1
    MOV   AX,1403H
    XLAT
     ⋮
```

6. (2 分)执行下列指令后,(BX)=?

```
      AND AX, 0
      MOV BX,0001H
      MOV CX,10
NEXT: ADD AX,BX
      INC BX
      LOOP   NEXT
```

7. (2 分)现有下列数据段:

```
DATA   SEGMENT
  DA1   DW   123H,48H,0AB00H
  DA2   DB   12H,34H,56H,0ABH
        DB   $ - DA2
  BUF1  DB   10H DUP(1,2,3)
DATA   ENDS
```

下列程序段执行后,寄存器 BX 的内容是什么?

```
MOV BX,DS:[0006]
AND BX,0FFH
ADD BX,[BX]
```

8. (2 分)下列程序执行后,写出以 DA2 为首地址的前三个字节单元的内容(也可画示意图)。

```
DATAS SEGMENT
  DA1 DB 'ABCDEFGHIJ'
  DA2 DB 10 DUP(?)
DATAS ENDS
CODES SEGMENT
   ASSUME CS:CODES,DS:DATAS,ES:DATAS
START:MOV AX,DATAS
```

```
MOV DS,AX
MOV ES,AX
LEA SI,DA1
ADD SI,3
LEA DI,DA2
MOV CX,3
CLD
REP MOVSB
    ⋮
```

五、程序填空(15分)

1.(6分)计算2+4+6+…+40共20个偶数的累加和,结果存放在SUM中。

```
DSEG   SEGMENT                    ;数据段
  SUM   DW   ?
DSEG   ENDS
CODE   SEGMENT                    ;代码段
    ASSUME CS:CODE,DS:DSEG
START:  MOV AX,DSEG
        MOV DS,AX
        MOV CX,20                 ;CX为计数器
        MOV AX,0
        MOV BX,2
NEXT:   ______(1)______           ;累加
        INC BX
        INC BX
        LOOP NEXT
        ______(2)______
        MOV   AH,4CH
        INT   21H
CODE    ENDS
      END   START
```

2.(9分)从键盘输入一个小写字母,将其转换成相应的大写字母并显示出来;若输入的是大写字母,则原样输出;若输入的是非英文字母,则结束程序(提示:大写字母的ASCII码比相应的小写字母的ASCII码值小。二者换算关系为:大写字母的ASCII码=小写字母的ASCII码-20H)。

```
CODE SEGMENT
    ASSUME CS:CODE
START:  MOV AH,01H
        INT 21H                   ;1号DOS功能调用
        CMP AL,'a'
        JB UPPER                  ;若小于,则转移到UPPER
        CMP AL,'z'
        JA STOP                   ;若大于,则属于非英文字母,转移到STOP
        ______(1)______           ;小写字母转换成大写字母
        MOV DL,AL
        JMP OUTPUT
UPPER:  CMP AL,'A'
```

```
        JB STOP
        CMP AL,'Z'
        JA STOP
        MOV DL,AL
OUTPUT: ____(2)____        ;显示输出
        ____(3)____
STOP:   MOV AH,4CH
        INT 21H
CODE   ENDS
    END START
```

六、程序设计(20 分)

1. (10 分)假设在 BLOCK 开始的缓冲区中存放有 30 个字符,包含其他非英文字符。编程实现：去掉非英文字符,然后进行大小写转换。要求：对其中的小写字母,将其转换成相应的大写字母输出；若是大写字母,则原样输出(不妨将数据定义为'1234efghIJKLmnOPQRsTUVw$#@abGH')。

2. (10 分)对一组字节无符号数进行比较,找出最小数,并将其存入 MIN 字节单元。数据段定义如下：

```
    ⋮
BUFFER  DB 10H,12H,3BH,43H,60H,0CH,32H,0AH,47H,1FH,32H,3DH
COUNT   EQU   $ - OFFSET BUFFER
MIN   DB ?
```

模拟试题三

一、单选题(15 分)

1. 堆栈中数据的存取方式是(　　)。
   A. 先进先出　　B. 后进先出　　C. 随机存取　　D. 顺序存取
2. 下面各指令中,书写正确的是(　　)。
   A. MOV　AH,100　　B. CMP　[BX],[BP]
   C. SUB 0200H,0100H　　D. ADC　AH,DX
3. 下列指令属于立即数寻址的是(　　)。
   A. MOV　AX,[2000H]　　B. MOV　DX,0ABCH
   C. MOV　CX,[BX+5]　　D. MOV　[1000H],BL
4. 下面的解释正确的是(　　)。
   A. 伪指令在汇编后产生相应的机器代码
   B. 汇编语言程序可直接执行
   C. 高级语言程序可直接执行
   D. 机器指令是可执行指令
5. 若定义"BUF DB 1,2,5 DUP(3,4)",则为变量 BUF 分配的内存单元数是(　　)字节。
   A. 4　　B. 7　　C. 12　　D. 8

6. 要将两个字符 A、B 顺序存放到存储单元 INPUT 中，正确的数据定义语句是(　　)。

A. INPUT DW 'BA'　　B. INPUT DW 'AB'

C. AB DB 'INPUT'　　D. AB DW 'INPUT'

7. 执行 DIV BX 指令后，存放商的寄存器是(　　)。

A. AL　　B. AH　　C. AX　　D. BX

8. 在默认情况下，指令“MOV [BX+5],AL”中的目的操作数使用(　　)段寄存器寻址。

A. CS　　B. DS　　C. SS　　D. ES

9. 若 AX=1978H，CL=4，则执行“SHR AX,CL”后，寄存器 AX 的内容是(　　)。

A. 9781H　　B. 9780H　　C. 0197H　　D. 8197H

10. 设 DS=1000H，ES=2000H，BX=3000H，指令“ADD AL,[BX]”的源操作数的物理地址为(　　)。

A. 3000H　　B. 4000H　　C. 13000H　　D. 23000H

11. 在顺序结构程序设计中，不可能使用的指令是(　　)。

A. 数据传送指令　　B. 算术运算指令　　C. 逻辑运算指令　　D. 转移指令

12. 设 AL=0AH，执行“XOR AL,0FH”指令后，AL 的内容是(　　)。

A. 0AH　　B. 0FH　　C. 50H　　D. 05H

13. 下面(　　)指令执行后，有可能改变 AL 寄存器内容。

A. AND AL, BL　　B. CMP AL, BL

C. OR AL, AL　　D. TEST AL, 02H

14. 当两个无符号数进行比较时，执行 JA NEXT 指令表示满足(　　)条件时转移到 NEXT 处。

A. CF=0 且 ZF=0　　B. CF=1 且 ZF=0

C. CF=0 且 ZF=1　　D. CF=1 且 ZF=1

15. 下面解释正确的是(　　)。

A. 指令“XOR AX,AX”执行后，AX 内容不变，只影响标志位

B. 指令“AND AL,80H”执行后，将 AL 的最高位置 1，其余各位置 0

C. 指令“NOT AX”执行后，AX 一定为 0

D. 指令“OR DX,1000H”执行后，将 DX 最高位置 1，其余各位不变

二、填空题(16 分)

1. 若将字数据 1234H 存放在以 21000H 开始的存储单元中，则 21000H 单元的内容为(__1__)，21001H 单元的内容为(__2__)。

2. 8086CPU 有(__3__)根地址线，因此其可寻址的存储空间为(__4__)。但是，所有可用来存放地址的寄存器都是(__5__)位的，无法直接提供实际的物理地址，所以采用了存储器地址分段的办法解决。

3. 存储器逻辑段分为 4 种类型，分别是代码段、(__6__)、(__7__)、(__8__)。编写程序时，必须有代码段，其他逻辑段可以根据实际需要选择。

4. 若运算结果为负数，则标志位 SF 是(__9__)。

5. 若定义 DA1 DW 4142H，则执行“MOV AL,BYTE PTR DA1”指令后，AL 的内容

为(　10　)。

6. 在串传送指令中,源串和目的串的偏移地址分别由(　11　)寄存器和(　12　)寄存器给出。

7. 若当前(AX)=7538H,标志位 CF=1,则执行“ADD AX,2345H”指令后,AL 中的内容为(　13　);若再接着执行一条 DAA 指令,则 AX 中的内容为(　14　)。

8. 调用子程序通常使用(　15　)指令,对于段内直接近调用,该指令执行后压入堆栈的内容是(　16　)字节。

三、简答题(14 分)

1. (8 分)按下列各小题的要求写出相应的一条汇编语言指令。

(1) 寄存器 AX 和立即数 2013H 相加,结果存入 AX 寄存器中,不考虑进位。

(2) 将寄存器 BX 和 CX 内容互换。

(3) 以 SI 和位移量 20H 作寄存器相对寻址,将该单元中的内容传送到 CX。

(4) 将 AH 中的数据低四位清零,高四位保持不变。

2. (6 分)写出完成下列操作的伪指令语句。

(1) 将字符串“HELLO”存放在 DA1 为首地址的存储单元中。

(2) 将字数据 6728H、5134H 存放在字变量 DA2 为首地址的存储单元中。

(3) 在 DA3 为首地址的字节存储单元中,连续存放数据:28、16 个(5,6),并预留 30 个单元(不存入数据)。

四、程序分析(20 分)

1. (6 分)若(SP)=2017H,(AX)=3040H,(BX)=5060H,则:

(1) 执行 PUSH AX 及 PUSH BX 后,(SP)=?

(2) 再接着执行 POP AX 及 POP BX 两条指令后,(AX)=? (BX)=?

2. (2 分)执行下列指令后,(DS)=?

```
MOV  AX,1302H
MOV  DS,AX
```

3. (2 分)执行下列指令后,(AX)=?

```
MOV  AX,2763H
MOV  AH,AL
```

4. (2 分)执行下列指令后,(AL)=?

```
MOV  AL,35H
MOV  BL,21H
ADD  AL,BL
```

5. (2 分)执行下列指令后,(AX) = ?

```
BUFFER DB 21H,52H,10H,20H,30H,40H,50H
        ⋮
    LEA BX,BUFFER
    INC BX
    MOV AX,[BX]
    SAL AX,1
```

6. (2 分)执行下列指令后,(AL) = ?

```
      MOV   AL,0
      MOV   BL,1
      MOV   CX,5
LP1:  ADD    AL,BL
      INC   BL
      LOOP  LP1
```

7. (4 分)程序段如下:

```
DA1   DB   'ABCDE12345'
DA2   DB   10   DUP(?)
        ⋮
       MOV   CX,000AH
       MOV   SI,OFFSET DA1
       MOV   DI,OFFSET DA2
NEXT:  MOV   AL,[SI+9]
       MOV   [DI],AL
       DEC    SI
       INC    DI
       LOOP   NEXT
        ⋮
```

(1) 写出以 DA1 为首地址的 10 个字节单元的内容(可画示意图)。

(2) 程序执行完后,写出以 DA2 为首地址的存储单元的内容(可画示意图)。

五、程序填空(15 分)

1. (9 分)从键盘输入一个字母,然后找出它的前导字符和后续字符(如 B 的前导字符为 A,后续字符为 C),并顺序显示这三个字符。不考虑输入有效性问题。

```
CODE   SEGMENT
  ASSUME CS:CODE
START: MOV AH,01H
       INT 21H
       ___(1)___
    ;换行
       MOV AH,02H
       MOV DL,0DH
       INT 21H
       MOV DL,0AH
       INT 21H
    ;顺序输出三个字母
       MOV DL,BL
       MOV AH,02H
       DEC DL
       INT 21H
       ___(2)___
       INT 21H
       INC DL
       INT 21H
       ___(3)___
```

```
        INT   21H
CODE    ENDS
   END   START
```

2.（6 分）下列程序是判断两个无符号字数据 X、Y 的大小，当 X＞Y 时执行 X－Y；当 X＜Y 时执行 Y－X；当 X＝Y 时执行 X＋Y，其运算后的结果存放在字变量 W 中。请在程序的空格处填上适当的指令。

```
          ⋮
        MOV AX,X
        MOV BX,Y
        CMP AX,BX
        JG L1                 ;若 X>Y,则转至 L1
        JB L2                 ;若 X<Y,则转至 L2
          (1)
        JMP EXIT
L1:     SUB AX,BX
        JMP EXIT
L2:     XCHG AX,BX
          (2)
EXIT:   MOV W,AX
          ⋮
```

六、程序设计（20 分）

1.（10 分）以 STR1 为起始地址的存储单元存有一个以'$'为结束标志的字符串（设长度＜255），测试该字符串的长度（不包含结束标志'$'），并将串长存入 STR1 单元，原字符串依次向后移一个单元存放。

2.（10 分）在数据段以 BLOCK 为首地址的存储区中，存放了 30 个非零字节数据。编程实现：分别统计正数和负数的个数，并将统计结果在屏幕上以十六进制的形式显示出来（注：数据段内容和显示格式信息自定）。

模拟试题一参考答案

一、单选题

1. A　2. C　3. A　4. D　5. B　6. A　7. C　8. C　9. D　10. C　11. B　12. D　13. C　14. D　15. A

二、填空题

(1) 2068H　(2) 32068H　(3) ES　(4) SS　(5) BX　(6) CX　(7) DX　(8) BX　(9) 字节　(10) 3824H　(11) 1　(12) 4154H　(13) 7FH　(14) 85H　(15) 1360H　(16) 3516H

三、简答题

1. (1) ADD AX,DX

(2) MOV CX,BX

(3) JZ NEXT

(4) ADD AX,AX 或 SAL AX,1

2. (1) DA1 DB 34H,79H

(2) DA2 DB 28H,67H,34H,5AH

3. 寄存器类型不匹配,可以改成：MOV AL,DL 或其他形式使之匹配。

四、程序分析

1. (1) AX=0100H (2) AX=4C2AH (3) 3412H

2. BX=7432H

3. AX=5858H

4. AL=6DH,CF=0

5. AL=16H

6. CX=0008H,AL=0BH

7. (1) 'ABCDEFGHMN' (2) 'DEF'

五、程序填空

1. (1) AND AL,0FH 或 SUB AL,30H

(2) INC DI

2. (1) CMP AL,[SI] (2) MOV AL,[SI] (3) MOV MAX,AL

六、程序设计

1. 参考程序：

```
DATA SEGMENT
   Message db 'Please input N: $ '
DATA ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       MOV AH,9
       MOV DX,OFFSET Message
       INT 21h
       MOV AH,1
       INT 21H
       CMP AL,'3'
       JB START
       CMP AL,'9'
       JA START
       SUB  AL,30H
       MOV CL,AL
       MOV CH,0
       MOV  AH,2
       MOV DL,0DH
       INT 21H
       MOV DL,0AH
       INT 21H
L1:    MOV DL, '*'
       INT 21H
       LOOP L1
```

```
        MOV DL,0DH
        INT 21H
        MOV DL,0AH
        INT 21H
        MOV AH,4CH
        INT 21H
CODE ENDS
    END START
```

2. 参考程序：

```
DATA SEGMENT
    GRD  DB 67,78,100,87,89,65,20,50,60,70
         DB 55,90,80,86,64,93,10,20,30,40
         DB 97,88,82,75,77,74,70,80,90,100
         DB 10,20,30,40,50,60,70,80,90,100
COUNT EQU $ - GRD
    SUM DW ?
DATA ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:DATA
START:MOV AX,DATA
      MOV DS,AX
      MOV SI,OFFSET GRD
      MOV CX,COUNT
      XOR AX,AX                 ;累加和单元清零
AA:   ADD AX,[SI]               ;求累加和
      INC SI                    ;地址指针指向下一个成绩
      LOOP AA
      MOV SUM,AX                ;存总分
      MOV AH,4CH
      INT 21H
CODE ENDS
END START
```

模拟试题二参考答案

一、单选题

1. C　2. A　3. D　4. A　5. D　6. C　7. D　8. B　9. C　10. C. 11. B
12. A　13. B　14. A　15. B

二、填空题

(1) 23451H　(2) 2B451H　(3) 64KB　(4) 16　(5) AH　(6) AL　(7) DL　(8) 17H
(9) 20H　(10) 0H　(11) 0685H　(12) 4192H　(13) 8EH　(14) 94H　(15) 1724H
(16) 4325H

三、简答题

1.（每小题 2 分，共 6 分）

(1) SUB AX,1024H；(2) MOV DL,BH；(3) JS NEXT

2.（每小题2分，共4分）

（1）MOV BX,OFFSET BUF1 （2）MOV CX,BUF2+4；或采用寄存器间接寻址/相对寻址

3.（4分）DA1 DW 'BA', 'DC'

DA2 DW 3412H,7856H

四、程序分析

1.（1）寄存器相对寻址，PA=(DS)*10H+(SI)+5 （2）寄存器间接寻址，PA=(SS)*10H+(BP) （3）基址变址寻址，PA=(DS)*10H+(BX)+(SI)

2. BX=7932H

3. AX=3369H

4. AL=00H，CF=1

5. AX=1412H

6. BX=000BH

7. 0314H

8. 'DEF'

五、程序填空

1.（1）ADD AX,BX

（2）MOV SUM,AX

2.（1）SUB AL,20H 或 SUB AL,'a'-'A' （2）MOV AH,02H （3）INT 21H

六、程序设计

1. 参考程序：

```
DATA   SEGMENT
   BLOCK DB '1234efghIJKLmnOPQRsTUVw$#@abGH'
DATA   ENDS
CODE   SEGMENT
   ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       MOV SI,OFFSET BLOCK
       MOV CX,30
NEXT:  MOV DL,[SI]
       CMP DL,'a'
       JB UPPER
       CMP DL,'z'
       JA OTHER
       SUB DL,'a'-'A'
       JMP DISP
UPPER: CMP DL,'A'
       JB OTHER
       CMP DL,'Z'
       JA OTHER
DISP:  MOV AH,02H
       INT 21H
OTHER: INC SI
```

```
        LOOP NEXT
        MOV AH,4CH
        INT 21H
CODE   ENDS
     END START
```

2. 参考程序:

```
DATA    SEGMENT
  BUFFER  DB 10H,12H,3BH,43H,60H,0CH,32H,0AH,47H,1FH,32H,3DH
  COUNT   EQU   $ - OFFSET BUFFER
  MIN     DB ?
DATA    ENDS
CODE   SEGMENT
   ASSUME  CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV SI,OFFSET BUFFER
        MOV CX,COUNT
        MOV AL,[SI]
        INC SI
        DEC CX
SEARCH: CMP AL,[SI]
        JB NEXT
        MOV AL,[SI]
NEXT:   INC SI
        LOOP SEARCH
        MOV MIN,AL
        MOV AH,4CH
        INT 21H
CODE    ENDS
        END   START
```

模拟试题三参考答案

一、单选题

1. B　2. A　3. B　4. D　5. D　6. A　7. A　8. B　9. C　10. D　11. D　12. D
13. A　14. C　15. B

二、填空题

(1) 34H　(2) 12H　(3) 20　(4) 1MB　(5) 16　(6) 数据段　(7) 附加段
(8) 堆栈段(6、7、8 可互换)　(9) 1　(10) 42H　(11) SI　(12) DI　(13) 7DH
(14) 9883H　(15) CALL　(16) 2

三、简答题

1.(每小题 2 分,共 8 分)

(1) ADD AX,2013H　(2) XCHG BX,CX　(3) MOV CX,[SI+20H]

(4) AND AH,0F0H

2. (每小题 2 分,共 6 分)

(1) DA1　DB 'HELLO'　(2) DA2　DW 6728H, 5134H

(3) DA3　DB 28, 16 DUP(5,6), 30 DUP(?)

四、程序分析

1. (SP)=20BH,(AX)=5060H,(BX)=3040H

2. DS=1302H

3. AX=6363H

4. AL=56H

5. AX=20A4H

6. AL=0FH

7. (1) 从 DA1 开始顺序存储,每个单元存一个字符。

'A','B','C','D','E','1','2','3','4','5'

(2) 从 DA2 开始顺序存储,每个单元存一个字符,字符顺序与 DA1 相反。

'5','4','3','2','1','E','D','C','B','A'

五、程序填空

1. (1) MOV BL,AL　(2) INC DL 或 MOV DL,BL　(3) MOV　AH,4CH

2. (1) ADD AX,BX　(2) SUB AX,BX

六、程序设计

1. 参考程序:

```
DSEG   SEGMENT
   STR1   DB 'ABCDEFGHIJK123456789 $ '
          DB ?
DSEG   ENDS
CSEG   SEGMENT
   ASSUME  DS:DSEG,CS:CSEG
START: MOV AX,DSEG
       MOV DS,AX
       LEA DI,STR1
NEXT:  MOV AL,[DI]
       CMP AL,' $ '
       JZ  NEXT2
       INC DI
       INC BL
       JMP NEXT
       INC DI
       MOV CL,BL
       MOV CH,0
NEXT2: MOV AL,[DI]
       MOV [DI + 1],AL
       DEC DI
       LOOP NEXT2
       MOV STR1,BL
       MOV AH,4CH
       INT 21H
```

```
CSEG  ENDS
   END  START
```

2. 参考程序：

```
DISPHEX MACRO  BLOCK
               LOCAL L1,L2
               MOV  BL,BLOCK
               MOV  DL,BLOCK
               MOV  CL,4
               SHR  DL,CL
               CMP  DL,0AH
               JB  L1
               ADD DL, 7
L1:            ADD  DL, 30H
               MOV AH, 2
               INT  21H
               MOV  DL, BL
               AND  DL,0FH
               CMP DL, 0AH
               JB  L2
               ADD DL, 7
L2:            ADD  DL, 30H
               MOV  AH, 2
               INT 21H
               ENDM
NEWLINE MACRO
               MOV DL,0DH
               MOV AH,2
               INT 21H
               MOV DL,0AH
               INT 21H
               ENDM

DATA SEGMENT
   BLOCK  DB  -110,26,35,-46,3,-88,-46,38,67,20
          DB 3,65,22,1,-7,-34,6,2,-7,10
          DB -71,-15,32,51,43,-15,-37,90,31,25
   DA1    DB  0
   DA2    DB  0
   PlusMsg   DB 'The number of positive numbers: $ '
   MinusMsg  DB 'The number of negative  numbers: $ '
DATA  ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA
START:MOV AX,DATA
      MOV DS,AX
      MOV BX,OFFSET BLOCK
      MOV CX,30
NEXT: MOV AL,[BX]
      AND AL,AL
```

```
        JNS L1
        INC DA2
        JMP L2
L1:     INC DA1
L2:     INC BX
        LOOP NEXT
        MOV AH,9
        MOV DX,OFFSET PlusMsg
        INT 21H
        DISPHEX DA1
        MOV DL, 'H'
        MOV AH,2
        INT 21H
        NEWLINE
        MOV AH,9
        MOV DX,OFFSET MinusMsg
        INT 21H
        DISPHEX DA2
        MOV DL, 'H'
        MOV AH,2
        INT 21H
        MOV AH,4CH
        INT 21H
CODE ENDS
    END START
```

参 考 文 献

[1] 于春凡，朱耀庭. IBM-PC 及长城 0520(INTEL 8088/8086)宏汇编语言程序设计[M]. 天津：南开大学出版社，1990.
[2] 沈美明，温冬婵. IBM-PC 汇编语言程序设计[M]. 2 版. 北京：清华大学出版社，2001.
[3] 朱耀庭，董焕芝，高飞. 汇编语言程序设计[M]. 北京：清华大学出版社，2013.
[4] 郑晓薇. 汇编语言[M]. 2 版. 北京：机械工业出版社，2016.
[5] 余朝琨. IBM-PC 汇编语言程序设计[M]. 北京：机械工业出版社，2008.
[6] 赵树升，杨建军. DOS/Windows 汇编语言程序设计教程[M]. 北京：清华大学出版社，2005.
[7] 张坤. 汇编语言实验教程[M]. 北京：清华大学出版社，2008.
[8] 龚沛曾，杨志强. 大学计算机[M]. 6 版. 北京：高等教育出版社，2013.
[9] K R Irvine. Assembly Language for Intel-Based Computers[M]. 影印版. 5th ed. 北京：清华大学出版社，2009.
[10] 家民软件专区，http://www.jiaminsoft.com/.

图书资源支持

感谢您一直以来对清华版图书的支持和爱护。为了配合本书的使用，本书提供配套的素材，有需求的用户请到清华大学出版社主页（http://www.tup.com.cn）上查询和下载，也可以拨打电话或发送电子邮件咨询。

如果您在使用本书的过程中遇到了什么问题，或者有相关图书出版计划，也请您发邮件告诉我们，以便我们更好地为您服务。

我们的联系方式：

地　　址：北京海淀区双清路学研大厦 A 座 707

邮　　编：100084

电　　话：010－62770175－4604

资源下载：http://www.tup.com.cn

电子邮件：weijj@tup.tsinghua.edu.cn

QQ：883604（请写明您的单位和姓名）



扫一扫

资源下载、样书申请

新书推荐、技术交流

用微信扫一扫右边的二维码，即可关注清华大学出版社公众号“书圈”。